DUDLE[illegible]
NAV[illegible]CHOOL
MONTER[illegible]3101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**An Automated
Ada Physical Source
Line Counter**

by

Kevin J. Walsh

September 1993

Thesis Advisor: Timothy J. Shimeall

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1993	Master's Thesis

4. TITLE AND SUBTITLE
An Automated Ada Physical Source Line Counter (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Walsh, Kevin John

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified/Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Tools to count lines of code have not been standardized or automated in a flexible fashion. This lack of flexibility can lead to ambiguous interpretations of the size of software modules, especially when the person performing the measurement does not use the method or rules expected by the person requesting the measurement. The Software Engineering Institute (SEI) Framework for Size Measurement provides a basis for flexible design of software measurements. The SEI framework describes measurements using nine attributes. This Framework is part of recently proposed DoD guidelines for software process measurement. The problem that this thesis addresses is how to implement the SEI Framework for Size Measurement to flexibility count lines of the code in Ada software. The approach is to build an automated Ada Physical Source Line Counter that measures Ada source files and generates the appropriate reports. The tool works as follows: the user defines the measurement constraints to the tool, which calls an Ada parser to generate counts to be included in user-specified reports. The result is a program that takes user requests and Ada source files and produces measurement reports as output. This program fully captures the flexibility of the SEI framework along five of the nine measurement attributes.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software Metrics, Source Lines of Code, Ada	286
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

An Automated Ada Physical Source Line Counter

by

Kevin J. Walsh
Major, United States Army
B.S.C.S., Youngstown State University, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

ABSTRACT

Tools to count lines of code have not been standardized or automated in a flexible fashion. This lack of flexibility can lead to ambiguous interpretations of the size of software modules, especially when the person performing the measurement does not use the method or rules expected by the person requesting the measurement. The Software Engineering Institute (SEI) Framework for Size Measurement provides a basis for flexible design of software measurements. The SEI framework describes measurements using nine attributes. This Framework is part of recently proposed DoD guidelines for software process measurement.

The problem that this thesis addresses is how to implement the SEI Framework for Size Measurement to flexibility count lines of the code in Ada software.

The approach is to build an automated Ada Physical Source Line Counter that measures Ada source files and generates the appropriate reports. The tool works as follows: the user defines the measurement constraints to the tool, which calls an Ada parser to generate counts to be included in user-specified reports.

The result is a program that takes user requests and Ada source files and produces measurement reports as output. This program fully captures the flexibility of the SEI framework along five of the nine measurement attributes.

Thesis
W222823
c.1

TABLE OF CONTENTS

I. INTRODUCTION 1
A. BACKGROUND 1
1. Size 2
2. Personnel/Effort 2
3. Computer Use 2
4. Schedule Progress 3
5. Requirement and Design Progress 3
6. Testing Progress/Quality 3
7. Incremental Release Content 3
8. Complexity 4
B. USERS OF SOFTWARE METRICS 4
C. WHY COUNTERS AT ALL? 6
1. Automated versus Manual Counters 6
2. Counting versus Reporting versus Tracking 6
D. PROBLEM DESCRIPTION 8
E. OVERVIEW OF THESIS 8
II. SEI FRAMEWORK 9
A. DEFINITION OF FRAMEWORK 9
1. Attributes 10
2. Values 10
3. Reports 11
B. APPLICATIONS OF FRAMEWORK 11
1. Software Size Measurement: 11
2. Software Effort and Schedule Measurement: 12
3. Software Quality Measurement: 12
C. AUTOMATION OF SIZE MEASURE 12

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

D. PREVIOUS SIZE CALCULATORS 13
1. DOS Version of the SEI Framework on Size 13
2. PC-Metric for Pascal 13
E. SUMMARY 14
III. TOOL DESIGN 15
A. ATTRIBUTES SUPPORTED 15
1. Statement Type 15
2. How Produced 16
3. Origin 16
4. Usage 16
5. Delivery 17
6. Development Status 17
7. Clarifications (General and Ada Specific) 17
B. ATTRIBUTES NOT SUPPORTED 18
1. Functionality 18
2. Replications 18
C. DEFAULT REPORTS/OUTPUT 18
1. Report A 19
2. Report B 19
3. Report C 19
4. Report D 20
5. Report E 20
D. USER-DEFINED REPORTS/OUTPUT 21
E. DATA STRUCTURES 22
1. Variables used for every report 22
2. Size Attributes 23
3. Five Dimensional Arrays 24
4. Priority Arrays 24

5. Flags Array 25
6. Current Settings Record 25
7. Checklist Variables 25
F. OVERVIEW 26
G. USER INTERFACE 27
H. PARSER 32
I. REPORT GENERATOR 34
J. SUMMARY 34
IV. TOOL USAGE 36
A. INTRODUCTION 36
B. REQUIREMENTS 36
1. Hardware 36
2. Software 36
3. Input 36
4. Legal Ada Syntax 36
C. LIMITATIONS 37
1. Package Conflicts 37
2. Coding Style 37
D. COMMAND LINE INVOCATION 38
E. EXTENDED EXAMPLE 38
1. Sample Application 38
2. User Interface 39
3. Statement Processing 42
a. Executable Statements 42
b. Declaration Statements 44
c. Comments on Own Line 45
d. Banner Comments 46
e. Blank Lines 46

f. OUTPUT 47
F. SUMMARY 53
V. SUMMARY AND CONCLUSIONS 55
A. RESEARCH SUMMARY 55
B. RECOMMENDATIONS 56
APPENDIX A USER MANUAL 58
A. REQUIREMENTS 58
1. Hardware 58
2. Software 58
3. Input 58
4. Legal Ada Syntax 60
B. LIMITATIONS 61
1. Package Conflicts 61
2. Coding Style 61
C. COMMAND LINE INVOCATION 62
D. USER INTERFACE 62
1. Push-buttons 63
2. Checkboxs 64
3. Radio-buttons 64
4. Labels and Text/Integer Keyin items 64
E. INTRODUCTORY PANEL 65
F. INPUT PANEL 65
G. ATTRIBUTE PANELS 66
H. CLARIFICATIONS (GENERAL and Ada) PANELS 72
I. GENERATE REPORT PANEL 72
J. QUIT PANEL 72
APPENDIX B. SOURCE CODE 77
APPENDIX C. EXTEND SAMPLE INPUT AND OUTPUT 244

LIST OF REFERENCES 271
INITIAL DISTRIBUTION LIST 273

LIST OF FIGURES

Figure 1 SEI Attributes supported and not supported 15
Figure 2 Written Description for Report A 19
Figure 3 Written description for report B 20
Figure 4 Written description for report C 20
Figure 5 Written description for report D 21
Figure 6 Written description for report E 21
Figure 7 Relationship of input file name and Ada source files to be measured 23
Figure 8 Attribute Statement Type values declared as an enumerated type 23
Figure 9 Declaration of five dimensional array 24
Figure 10 Declaration of Priority type array to track precedence levels 24
Figure 11 Declaration of FLAGS_TYPE_ARRAY 25
Figure 12 Example of FLAGS_TYPE_ARRAY values set to true 25
Figure 13 Overview of the Automated Ada Physical Source Line Counter 26
Figure 14 Overview of User interface 31
Figure 15 Example of Ada code executed inside of Ayacc 32
Figure 16 Example of rule to find blank lines 32
Figure 17 Example of rules added to Aflex 33
Figure 18 Example of code to recognize Special comments 34
Figure 19 Example of Ada.y input file to Ayacc 37
Figure 20 Example of two different coding styles 38
Figure 21 Contents of EXAMPLE_FILE 39
Figure 22 Example of text keyin fields 40
Figure 23 Example of marked checkboxs for selecting reports A - F 40
Figure 24 Example of customizing report F 41
Figure 25 Variables used during parsing of source files 42
Figure 26 Sample input code 44
Figure 27 Partial output of Report A 48
Figure 28 Partial output of Report B 49
Figure 29 Partial output of Report C 50
Figure 30 Partial output of Report D 51
Figure 31 Partial output of Report E 52
Figure 32 Partial output of Report F 54
Figure A-1 Special Comments 59
Figure A-2 Example of Ada.y input file to Ayacc 61
Figure A-3 Example of two different coding styles 62
Figure A-4 Example of Push-buttons 63
Figure A-5 Available Report Names 64
Figure A-6 Text Keyin fields and labels 65
Figure A-7 Statement Type Panel 67
Figure A-8 How Produced Panel 68
Figure A-9 Origin Panel 69

Figure A-10 Usage and Delivery Options Panel 70
Figure A-11 Development Status Panel 71
Figure A-12 Clarifications (general) Panel 73
Figure A-13 Clarifications (Ada) Panel 74
Figure A-14 Generate Report Panel 75
Figure A-15 Quit Panel 76

ACKNOWLEDGEMENTS

This project is not the work of one individual alone. There is a large number of people who contributed to this work both directly and indirectly. Although it would be impossible to acknowledge all of them, I would like to take this opportunity to thank the major player, Dr. Timothy J. Shimeall, who provided endless support and encouragement during this work. His guidance was invaluable. Additionally, I would like to thank MAJ Gaitros for his contribution in making this thesis more understandable. Thanks are also due to Mr. Bob Park and others at SEI for their previous work providing the basis for this thesis.

Bob Ordonio helped me see the light in numerous programming dark holes.

Finally, I would like to thank my wife, Susan, my son Sean and my daughter Sara, for their loving support and understanding during the many nights spent at the terminals in the Computer Science Lab.

Without the help and moral support of the above people and countless others, this work would not have been possible.

I. INTRODUCTION

A. BACKGROUND

From 1980 to 1985, software-related costs rose from 3 percent ($40 billion) of the U.S. gross national product to 5 percent ($228 billion). Software effects are increasing, due in part to: the decreasing cost of hardware, which has been cut in half every two years; the increasing speed and capacity of computers to new applications as computers do tasks that are either too complicated or too time-consuming to do manually. However, the systems that rely on software will only use software that is reliable, easy to use, and accomplishes the needs of the people using the systems. The software developer is tasked to ensure that software is delivered on time, under budget and meeting or exceeding the performance requirements. [BER 90]

Software is not just the code, but the entire set of documentation, operating procedures, test cases, and programs associated with a computer-based system. The goal of software engineering is to provide effective methods for producing software systems that meet the needs of the customer, while conforming to the customer's schedule and budget constraints. [BER 90]

Part of this process is to provide accurately-generated attributes to estimate cost and effort, then track the ensuing process against the estimate. The field of computer science does not currently quantify accurately its attributes. [BEI 90]

Beizer categorizes computer metrics into three groups, which are: Linguistic, Structural, and Hybrid. Linguistic metrics are measurements without regard to interpretation. For example, most counts of things are linguistic. The measurement of the number of unique operands or number of unique operators is linguistic. The order of either operands or operators is of little concern. The total number of operators or operands is the target, not where in the code or how they interact with each other or other parts of the code. [BEI 90]

Structural metrics are based upon the structural relations between objects within the program. Structural metrics measure the properties of the control flowgraphs of data flowgraphs. An example would be to count the number of links, number of nodes, and nesting depths. To accomplish this type of metric, the program would need to be interpreted. Hybrid metrics are some combination of linguistic and structural. [BEI 90]

There are many different measurements currently in use. Some measure the size of a project from the number of lines it contains, others base the size on the number of operators. There are measures for productivity, schedule, effort, quality, requirements designed, detailed designs, and use of availability of computer resources. Some commonly used metrics include, but are not limited to Size, Personnel, Computer use, Unit Progress, Schedule Progress, Design Complexity, Requirement and design Progress, Testing Progress, and Incremental release content. The next sections will briefly discuss some metrics available and the benefits of using them.

1. Size

The size measurement may reflect either the planned size or current or estimated size. One possible unit of measure is lines of code to be developed, modified, planned, and reused. This metric can help plan the total effort and schedule and measure productivity. For an example, an increase in the size of the software code, shows the project will require more resources of time, money and personnel.

2. Personnel/Effort

The personnel/effort measurement includes the estimated and current number of personnel working on the project. One possible unit of measure is time card hours. This metric measures productivity, and how staffing is effecting the planned schedule and cost.

3. Computer Use

The computer use measurement includes the estimated and actual percentage of the target system's hardware CPU, storage, and communication capacity. A possible unit

of measure is the CPU speed. This metric shows if the planned target system is capable of the current requirements and if any spare capacity exists for increases later.

4. Schedule Progress

Schedule progress measurement includes the estimated progress, measured as the ratio of the planned to the actual work done on the schedule. This metric can measure productivity and progress. For example, this measurement can show if the software development is meeting the scheduled requirements as laid out in the contract. This metric uses the standard cost-reporting data on software work packages completed under MIL-STD 2167A.

5. Requirement and Design Progress

The metric for requirement and design progress can track requirement documentation process. The measurement would detail the number of requirements, number of requirements documented, and the number of requirements scheduled to be documented. This metric is used in the specification and detailed design phases. This metric is another measure of progress and productivity.

6. Testing Progress/Quality

The measurement of progress and quality includes the planned and actual configuration items and completed system tests; number of new problem reports and opened or unresolved problem reports. These metrics can measure the progress in completing testing, the number of potential bugs found, and the speed of fixing them. These metrics can also estimate software quality, and the time needed to complete the tests.

7. Incremental Release Content

This metric, compares the estimated and actual release date with the estimated and actual components in each release. The metric measures the progress of a software project in relation of the estimated module releases, with the actual module releases. For example, if the number of actual modules increases for a planned release, is this as a result

of changing requirements or coding is ahead of schedule? Conversely, if the number of actual modules decreases for a planned release, is this a result of pressure to meet the published schedule dates, and coding is behind schedule?

8. Complexity

The most common design complexity metric is Cyclomatic complexity [NAS 90]. This metric indicates which parts of the software system that may be error prone or hard to maintain. For example, a software developer may use cyclomatic complexity on a piece of software such that, once a software module exceeds a level the module is reworked to bring the complexity level down if there is sufficient time and budget for the rework. Cyclomatic complexity may also provide a relative indication of where testing will be difficult, which aids in planning test efforts.

B. USERS OF SOFTWARE METRICS

Program managers and software developers are the main users of software metrics. Users can benefit from the use of software measurement throughout a software project.

Program managers use software metrics to help them estimate project costs, schedules and performance. A good estimate is the cornerstone of a successful project. To have a successful program, the manager needs to know how much money to spend, (cost), how much time to finish the program (schedule), and what the final specifications are (performance). When dealing with software projects, the manager needs to know how long it is going to take to develop the final product and at what cost. To do this effectively, a good, accurate, estimate early in the program's life cycle is needed. The estimate of the project size will affect the cost, schedule, and performance qualities of the project. For instance, if the project is estimated to have the functionality of A, B, and C; the project must be completed in one year; the project is estimated to have 100K lines of code; with five programmers; and no more than $500K. Management needs to ensure that resources are employed to complete the project on time or to make a decision to trade-off one resource for another. Management will also need to know during the life of the project the status of

the constraints placed on this project - is it falling behind or ahead of schedule, exceeding or within its budget.

Software metrics can help decide where the software project is in relation to the project completion and alert project managers, maintainers, testers of future problems, delays, and increased costs. During a large multi-year software development, there are usually only a few major milestones that must be met by the software developer. The major milestones do not allow for adequate management visibility of the entire process for a day to day or week to week development tracking. A lot happens between these milestones, and a more detailed picture is needed on a more frequent basis. In order for management to track the progress of a project, they will need to know a frequent basis, day-to-day, week-to-week, or month-to-month how the project is progressing along the planned route. Any deviations from this planned route will cost the development some precious resources down the line, either time or money or schedule. The sooner management can make an informed decision, the better. A good set of software metrics provides this information. However, clear guidance must be given on what data to collect and made available to users.

The use of metrics is still in the development stages with no existing standards to what to count or report. Reports within an organization may fluctuate from project to project, depending on the importance placed on metrics. This lack of standardization interferes with comparing of lessons learned from one project to another project.

Software projects, like any other projects contain a certain amount of risk. The use of inaccurate or paradoxical metrics increases that risk. The factors that contribute to this include lack of: standardization for languages; rules to compare across languages, such as Ada and C; accurate size estimations during the requirements phase, when this information is needed.

Software metrics are not a panacea, but merely a tool that can deliver response to management and the technical staff on various aspects of a software project. For example, when using a complexity metric, once a module has exceeded a certain predetermined

threshold, then remedial action might bring the complexity of the module back within allowed limits by dividing the code into smaller less complex modules. [SLI 87]

C. WHY COUNTERS AT ALL?

The use of size metrics (Source lines of code (SLOC) and logical source statements) include tracking, planning, budgeting, maintaining and estimating software projects. Software developers also use software size metrics to plan, control, and improve their product. [SEI-B 92]

1. Automated versus Manual Counters

Performing software measurements with a purely manual technique is expensive because of the extra-ordinary amount of time involved even for the simplest of measurements. In addition, performing any function manually may introduce errors because humans are fallible. To overcome these problems and provide the flexibility, consistency and reliability the process of making these measurements must be automated.

2. Counting versus Reporting versus Tracking

The smart software developer will get the most out of the information available from metrics. The metric numbers are the result of performing some measurements against a piece of software. These numbers represent the counting portion of the metric. The results will differ when the rules are changed. The displays of the outputs from the count have to deal with the reporting aspect of the metrics. The user requesting, the information need to be able to get the information in a form that he is expecting, and one that will be of use. Tracking refers to the ability to trace the progress of the software development project completely. During tracking, the manager can decide if the project is meeting the schedule.

The Department of Defense (DoD) is trying to deal with the effects of a reduced budget. As DoD rightsize itself, organizations are going to have to do more with less. The software community will be no different. DoD has set up some long term goals that include: reducing the life-cycle costs by a factor of two; reducing software problems rates by a

factor of ten; achieving new levels of DoD mission capability and interoperability via software.

These goals imply that DoD can apply some measure of where they are now, against where they are going, to know if they ever get there. However, no baseline has been set to measure progress toward these goals. DoD has teamed up with the Software Engineering Institute (SEI) to provide a part of the baseline that will be necessary to meet the above goals by the year 2000. Software metrics will provide the measurement from the baseline of the software development progress. [SEI-A 92]

SEI's task is to provide a core set of measurements for use within DoD software projects. In 1992, SEI published several frameworks accomplishing the goal. DoD agencies can use these frameworks to plan, monitor, and manage its software projects, both internal and contracted. These SEI framework documents outline how to: define what is to be measured (set a standard), by using a checklist; and to express clearly the results of those measurements, (provide unambiguous results) consistently.

The SEI' frameworks help management to answer several key questions. How big is the job? Can our staffs meet the added commitments? Can we deliver on schedule? How reliable is our project? Will the project meet fielding deadlines? Will the project meet the required specs? Will the project need more time and effort due too unplanned releases to fix detected bugs? The answers to these questions will help the software developer ensure the proper mix of personnel are available for the project and give an idea of where the project is in relation to the baseline.

SEI has concentrated on defining unambiguous measures for size, effort, schedule, and quality. It is SEI's objective to provide tools the project manager can use concerning project planning, project management and process improvement. Consistent measurements are crucial to the project manager.

D. PROBLEM DESCRIPTION

Tools to count lines of code have been around for a long time. However, these tools are not standardized, automated and can lead to misleading and ambiguous interpretations of the size of software modules, especially when the person performing the measurement does not use the method or rules expected by the person requesting the measurement.

One question addressed by this research looks at the ability to automate the SEI framework for Software Size Measurement. Introducing automation to most processes reduces the time required to complete the process since machines are inherently faster than man. If automating the SEI framework for Software Size Measurement provides more benefits compared to costs, then it may be of value to carry out the framework.

Another question addressed by this research pertains to the ability to provide the standardization of the SEI framework for Software Size Measurement. Introducing standardization to the measurement process will provide unambiguous, clear and consistent reports, that management can use to track their software development process.

A final question addressed by this research looks at how to provide the flexibility outlined in the SEI framework for Software Size Measurement. The SEI framework is designed to allow over eighty different values, while still maintaining the consistency and reliability of the counting tool.

E. OVERVIEW OF THESIS

Chapter II provides background information related to this thesis. Topics covered include an overview of the SEI frameworks on size, effort and schedule, and quality. Chapter III discusses the tool design. This includes each of the major parts of the tool, TAE, Ayacc, and Aflex, giving an overview, purpose and interaction with the other parts. Chapter IV describes how to use the tool. Chapter V provides a summary of conclusions and further work. Appendix A contains a user manual. Appendix B list the source code for the tool. Appendix C lists some sample inputs and outputs.

II. SEI FRAMEWORK

A. DEFINITION OF FRAMEWORK

A framework is "a structure to hold together or support something" [WAR 90]. SEI has published three frameworks in the area of software measurement. The three frameworks are concerned with size, effort and schedule, and quality. These frameworks are the result of years of work by several groups of software professionals. [SEI-A 92]

SEI does not propose that these frameworks as standards set in concrete, but to use them as a basis for collecting information concerning the development of software. These frameworks provide measurements that will lead to unambiguous and mutually exclusive reports. To achieve this, each SEI framework uses two criteria. The first is communication. A measurement is not useful if the report user does not understand the results, or the rules used to get them. The report needs to convey what was measured and what was not measured. The second criterion is repeatability. The measurement, when applied by others, should have the same results. Consistent report results provide confidence to the users. Users will not use a measurement that does not provide stable results. [SEI-A 92]

Each framework proposed by SEI has some common structures. They all use checklists and recording forms. The checklists provide the repeatability mechanism. Having a checklist filled out and on hand, any person performing the measurement knows what is to be measured. The recording forms provide this communication mechanism. The person reading the recording forms can see what was measured and what was not. [SEI-A 92]

Developers of software projects are looking at ways to help manage the entire software process as the software projects get larger and more complicated. The software developer goal is to produce code that is on time, reliable, and performs as the users requested. Software developers can add these frameworks to their toolbox and use them during the life cycle of a project. The use of measurements can help the software developer produce quality code on time and under budget. [SEI-A 92]

A problem with previous measurement was not doing the measurement, but communicating the result so they have meaning to the user. Important parts of the frameworks are the attributes of the measurement.

1. Attributes

An attribute is "something seen as belonging to or representing someone or something." [WAR 90] The attributes associated with the SEI frameworks provide insight, definition and characteristic of the software project being measured. [SEI-A 92]

The SEI checklist identifies the attributes that need to be measured to ensure the two criteria of communications and repeatability. The checklist shows what attributes are included or excluded for each report. To ensure that measurements are accurate and non-overlapping, SEI has carefully chosen the attributes so that they are orthogonal in nature. The attributes are the broad categories of the frameworks. Each attribute is made up of two or more values to provide a finer measurement. [SEI-A 92]

2. Values

The values for each attribute were chosen so that they are mutually exclusive of each other. The reason for this is to help eliminate misunderstandings that can result if the values for any one attribute are overlapping. Values are listed on the checklist form. The measurement user then fills out the form and either chooses each value as included in the measurement, or excludes the value from the measurement. Statement type is one attribute within the SEI framework for Size. The listed values for this attribute are: executable, declarations, comments on their own line, comments on line with source code, comments that are banners or nonblank spacers, empty comments and blank lines. This is not an exclusive list, users can add or change as they see fit. However, users need to ensure that changes or modifications to the values keep the mutually exclusive property. Changes that violate the mutual exclusion property could result in possible double counting of some values. [SEI-A 92]

3. Reports

After the checklist has been filled out, and the measurement performed, the next step is to express the results of the measurement in a way that can be understood and read by the people who use them. Each of the three frameworks that SEI published has an example of some predefined initial reports and the associated value settings. The reports for the SEI framework on Size are discussed in more detail in chapter three. A more detailed discussion of reports for the other two frameworks can be found in the Software Engineering Institute reports.

B. APPLICATIONS OF FRAMEWORK

1. Software Size Measurement:

The SEI framework for size provides two independent templates that can measure the size of software, physical source lines and logical source statements. The use of size measurements can be used by software project managers to plan, maintain, track, and estimate software projects. [SEI-B 92]

Another goal of SEI was to reduce ambiguities and misunderstandings in the different reports of software size. Without using a process similar to the SEI framework, reports containing statements like "our activity produced over 500K source lines of code" would be meaningless to everybody except the person performing the count. The reason this count is unclear is the fact that the statement does not tell the reader what was counted, what was not counted, rules used to perform the count, if the count included all software modules, or just newly developed modules. The user of the reports needs to understand the rules and methods used to perform the counting operations. SEI's framework on size provides for complete and explicitly defined measures for both physical source lines of code and logical source statements. [SEI-A 92]

The framework also provides for the ability to ensuring that the report received this month is consistent with the report received last month and the one that will be received a year from now. This consistency will allow users to gain insight into project trends, to

compare one project against others, and ultimately to make necessary corrections if needed in the total software development process. [SEI-A 92]

2. Software Effort and Schedule Measurement:

The framework for effort and schedule provides a starting point for building unambiguous measures that will help manage, software projects and processes. The SEI reports are an approach to gather information for defining and recording staff-hours and related schedule information. There are many reasons for collecting and using data for staff-hours, three of which are: to pay individuals (payable hours), to charge for hourly services (billable hours), and to use in productivity and quality studies (actual hours). [SEI-A 92]

3. Software Quality Measurement:

As with the other SEI frameworks, the goal of the quality framework is to provide the user the ability to obtain clear, non-overlapping and repeatable reports of software quality. The framework includes: the relationship of the discovery, reporting, and measurement of problems and defects; a set of measurable, orthogonal attributes for making the measurement descriptions exact and unambiguous; checklists for creating unambiguous and explicit definitions or specifications of software problem and defect measurements; examples of how to use the checklists to construct measurement specifications; and examples of measurements using various attributes of software problem reports and defects. The reports of software quality can help the user to estimate, plan, and track, the software development process. [SEI-D 92]

C. AUTOMATION OF SIZE MEASURE

There are several reasons for selecting the size measurement as the measurement to automate first. They are:

- Most of the historical data for cost models and project estimating are based on physical measures of source code size.
- Size measurements are easier to define and use.

D. PREVIOUS SIZE CALCULATORS

There are two size calculators that were looked at as examples of tools to count source line of code. One tool was a static prototype of the SEI framework, on DOS a platform. The other tool was a mature DOS product for Pascal programs, to calculate non-SEI size measures and other metrics.

1. DOS Version of the SEI Framework on Size

The version tested was a prototype of the SEI framework on size. The tool was not completely functional. The tool as tested only carried out the attribute's statement type and origin. The other attributes were scheduled to be set up at a later release date. The tool also did not allow for the operator to change the settings of the values for the two attributes supported. The settings for both attributes were set to "included." [CSC 92]

AdaSAGE provides the user interface to the tool. The interface was a copy of the checklist form provided by the SEI framework. The tool requires the user to type in the name of a source file. The source file will contain the names of Ada modules. The tool will write the results of the measurement to a file. [CSC 92]

A session at a recent software engineering conference included descriptions of tools developed in parallel with this thesis, but these were unavailable for examination. [SEI 93]

2. PC-Metric for Pascal

The PC-Metric for Pascal is a DOS-based tool that provides three separate measurement reports on Pascal programs. The three reports are the complexity report, the exception report and the error report. As a part of these reports, the PC-Metric for Pascal also provides information on the number of lines of code contained in the source file. However, the lines of code are just that, the total number of lines of Pascal code. No information on the statement types that were counted, how the code was produced, the origin of the code and so on. The method for obtaining the line count was to count the total number of lines in the source file. [SLI 87]

However, the PC-Metric for Pascal does provide some insight into the complexity, the exception and errors of the Pascal source file. Also included with the documentation, is a tutorial on software metrics. The tutorial covers how to develop metrics, metrics' accuracy, specific metrics and how to use metrics in software development. [SLI 87]

E. SUMMARY

This chapter has discussed the SEI framework, the building blocks of the framework, how the frameworks can be applied, and examples of some tools that provide physical source lines of code measurements. The SEI framework is designed to provide a flexible, automated measurement tool based on consistency and reliability. The two tools summarized above do not provide the flexibility of the SEI framework. The Automated Ada Physical Source Line Counter provides a flexible, automated tool that provides consistent and reliable results.

III. TOOL DESIGN

A. ATTRIBUTES SUPPORTED

The SEI framework on size details nine different attributes that are orthogonal in nature. The Automated Ada Source Line Counter implements six of the nine attributes. The following sections discuss the attributes in detail. (See Figure 1)

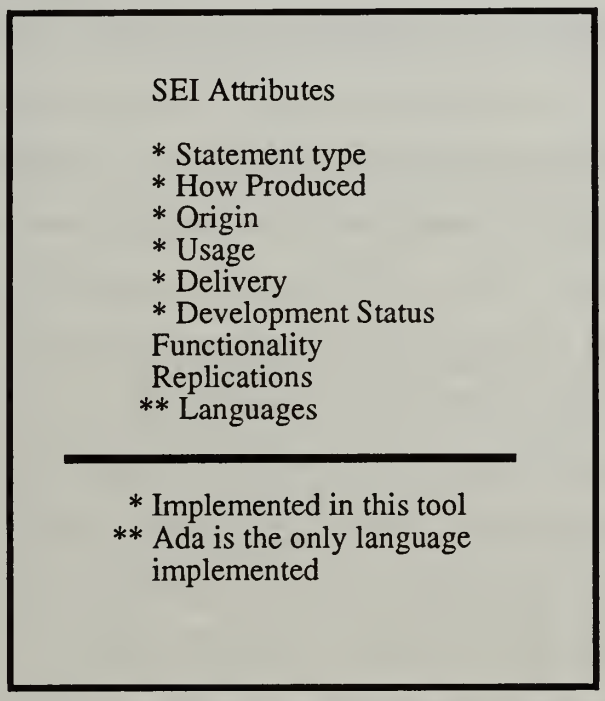

Figure 1 SEI Attributes supported and not supported

1. Statement Type

The statement type attribute distinguishes the source statements according to the function they perform. There are five basic types of statements, they are: executable, declarations, compiler directives, comments, and blank lines. Comments are further subdivided into: comments on their own lines, comments on lines by themselves, banner and nonblank spacers, and empty comments. The value of this attribute is determined for each physical source line of code during the parsing of the input files. [SEI-B 92]

2. How Produced

This attribute is used to identify the process by which the individual line of code was produced. This attribute is divided into six values, that include: programmed, generated with source code generators, converted with automated translators, copied or reused without change, modified, and removed. The value of this attribute is determined by special comments in the source code. The default value for this attribute is 'programmed'. Special comments are discussed in greater detail in PARSER on page 32. [SEI-B 92]

3. Origin

The attribute origin tracks the prior life, if any, of the product software. The origin attribute is divided into eleven values. The values are the following: new work; a previous version, build or release; commercial, off-the-shelf software (COTS), other than libraries; government furnished software (GFS), other than reuse libraries; another product; a vendor-supplied language support library (unmodified); a vendor-supplied operating system or utility (unmodified); a local or modified language support library or operating system; other commercial library; a reuse library (software designed for reuse); and other software component or library. The value of this attribute is determined by special comments in the source code. The default value for this attribute is 'new work'. [SEI-B 92]

4. Usage

The usage attribute makes the distinction of code developed as part of the software project and code not developed for the software project. The attribute usage is divided into the values in or as part of the primary product and external to or in support of the primary product. Code that is developed as part of the software project could have different costs associated with the development, maintenance, and testing of the code versus code that is developed for support of the code. For example, test drivers are not maintained or documented at the same level as the primary code. Distinguishing the differences is important for reports of productivity, quality, effort, and progress. The value

of this attribute is determined by special comments in the source code. The default value for this attribute is 'in or as part of the primary product'. [SEI-B 92]

5. Delivery

The delivery attribute distinguish between the form and destination of the source code. In this tool, delivery means delivered to the organization that will maintain the source code. The delivery attribute is divided into four values, that include: delivered as source; delivered in compiled or executable form, but not as source; under configuration control; and not under configuration control. For reports A through E, this value is set to 'delivered as source'. For report F, the tool user has the option to pick one of the four values of this attribute. The default value for report F is 'delivered as source'. [SEI-B 92]

6. Development Status

The development status attribute is used to mark the progress of the source code from the design phase to a finished product. The count of the various values of development status can provide insight into the development and integration workload yet to be accomplished. The attribute development status is divided into eight values, which are: estimated or planned; designed; coded; unit tests completed; integrated into components; test readiness review completed; software (CSCI) test completed; and system test completed. The value for this attribute is determined by special comments in the source code. The default value for this attribute is 'system tests completed'. [SEI-B 92]

7. Clarifications (General and Ada Specific)

The clarification attributes, both the general and Ada specific, aid in explaining the rules used to define the differences among the eight values of the attribute statement type. The general clarifications are for any language. The Ada specific clarifications deal only with Ada programming language issues. Each general and Ada specific clarification is associated with a unique statement type attribute value. For example, the default setting for counting a null statement, is to count the null statement as an executable. The

clarifications for reports A through E can not be changed. In report F, the user may change the clarifications rules from one of the attribute statement type values to another. [SEI-B 92]

B. ATTRIBUTES NOT SUPPORTED

There are two attributes of the SEI checklist that were not implemented in this tool. They are functionality and replications, which are discussed briefly in the following paragraphs.

1. Functionality

The attribute functionality deals with whether or not a line of source code is a functional part of the code or not. The attribute functionality is divided into two parts operative and inoperative. Inoperative is further divided into inoperative but functional (intentional dead code, reactivated for special purposes) and nonfunctional (unintentionally present). [SEI-B 92]

2. Replications

The attribute replication describes how to account for a software project's master source statements from its copies. There are four values for the attribute replications, which are: master source statements (originals); physical replicates of master statements, stored in the master code; copies inserted, instantiated, or expanded when compiling or linking; and postproduction replicates -- as in distributed, redundant, or reparameterized systems. [SEI-B 92] This attribute was not implemented in this tool, although combinations of the Ada Physical Source Line Counter with a differencing tool such as the unix DIFF might be useful [SUN 90].

C. DEFAULT REPORTS/OUTPUT

There are a total of six reports that this tool can generate. Five of the reports are defined by the SEI framework on Size. A sixth report is provided to allow the user to create, modify,

and use as they see fit. Appendix C contains examples of each reports output. Each report will be briefly discussed in the following paragraphs.

1. Report A

Report A is the basic definition for counting physical source lines of code. This report will give us information on the total noncomment and nonblank physical source lines of code. Report A explicitly spells out the rules to be used when comments are on the same lines as other source statements. The report also addresses all origins, stages of development and code that is integral to the product and external to the product, and forms of code production. Reports B through F build upon this basic definition. See Figure 2 for a written specification for this report. No data arrays are included in this report. [SEI-B 92]

> For the programming language Ada, measure and record these values:
> Total lines
> Individual totals for all values included

Figure 2 Written Description for Report A

2. Report B

Report B provides the capability for project tracking. Report B provides this information through the expanded use of the development status attribute. For example, report B will provide the progress of the software project through each of the production processes, the how produced attribute in comparison to the stage of development and the development status attribute. This is accomplished through periodic measurements using report B and comparing the results of the two-dimensional array. See Figure 3 for a written specification for this report. Appendix C contains an example of report B and the associated two-dimensional array. [SEI-B 92]

3. Report C

Report C is designed for the end of project data gathering. The results of this report can be used to improve future estimates and planning for future projects. This data

For the programming language Ada, measure and record these values:
Total lines
Individual totals for all values included
A two-dimensional array showing the number of lines
in each development status
for each production class

Figure 3 Written description for report B

would be collected at the end to help fine tune existing estimates and cost models and for the estimation of future software projects. Report C adds to report A by including the values of comments on their own lines and comments on lines with source code for the statement type attribute, the value removed for the how produced attribute, and a two-dimensional array, six by eight, containing the attributes statement type and how produced. See Figure 4 for a written specification for this report. [SEI-B 92]

For the programming language Ada, measure and record these values:
Total lines
Individual totals for all values included
A two-dimensional array showing the number of lines
in each statement type
for each production class

Figure 4 Written description for report C

4. Report D

This report measures reuse of software code. The data elements included in this report will help the user to quantify and interpret the amount of software reuse. This report can also be used to measure productivity and quality. Report D, in addition to the information for report A, asks for a two-dimensional array, six by eleven, containing the attributes of how produced and origin, and includes the value removed for the attribute how produced. See Figure 5 for a written specification for this report. [SEI-B 92]

5. Report E

Report E is the combination of report C and D. Combining the two reports can save resources including time, money, and paper. However, the trade-off for this report is

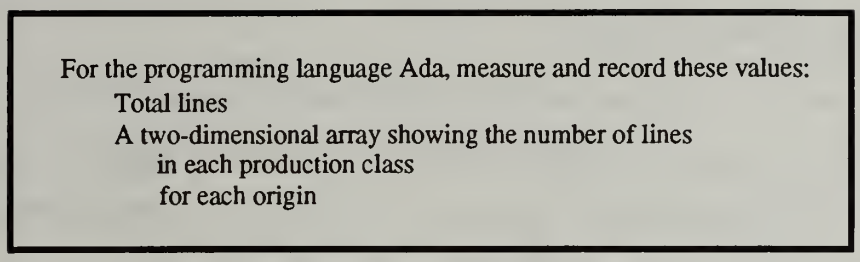

Figure 5 Written description for report D

the creation of a three-dimensional array. Three-dimensional arrays are harder to communicate to the user, especially using two-dimensional medians such as monitors and paper. Report C generated one two-dimensional array of six by eight. Report D generated one two-dimensional array of six by eleven. Report E on the other hand, generates eleven two-dimensional reports of six by eight. The higher number of arrays are required to facilitate the display of information that can be displayed on a terminal, written to an ASCII file, or printed on paper. A three-dimensional array is displayed by taking the third dimension of size eleven and creating one two-dimensional array of six by eight from the other two dimensions. See Figure 6 for a written specification for this report. [SEI-B 92]

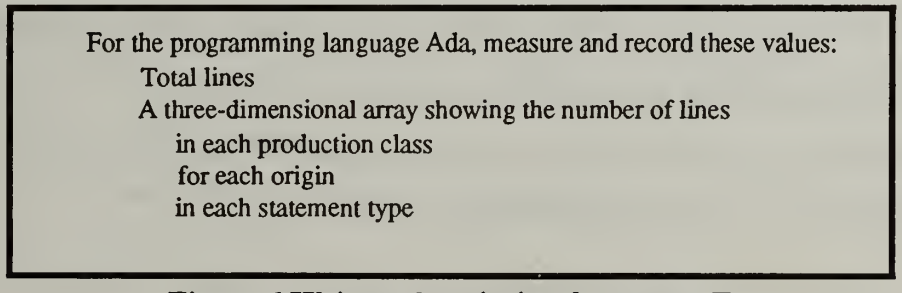

Figure 6 Written description for report E

D. USER-DEFINED REPORTS/OUTPUT

The tool enables the user the ability to create any unique report. This report is left up to the user of the tool to design and create according to requirements. The initial settings are the same as report A. For instance, the user would use report F to assign a different priority to the values of the statement type attribute.

E. DATA STRUCTURES

There are several key data structures used throughout the tool. The data structures are declared in the Transportable Applications Environment (TAE) global package because of the need for TAE generated Ada code to have the ability to set key variables used for every report and to change, as required, the user defined report.

The key elements of the data structure include: variables used for every report; the size attributes declared as enumerated types; one five-dimensional array to hold the source line count per reports; one one-dimension array to track the precedence levels of the statement type attribute; one two-dimensional array that maps the eight different statement type values to two boolean variables to track statements on lines; a record structure containing five fields, one field for each of the five dimensions and a record that contains all of the supported attribute values.

Several of these different data structures are repeated, one instance of each data structure for each possible report. Having the multiple instances of the data structure allows for up to six different reports to be generated at the same time, even though the Ada input files are only parsed once. A trade-off was made in favor of time to make repeated runs of the tool versus the extra storage space needed to generate all six reports during one run. A detailed discussion of each key data structure follows.

1. Variables used for every report

Every time that the Automated Ada Physical Source Line Counter is invoked, there are two pieces of information that must be entered by the user for the tool to run properly. The two pieces of information are the names of the input and out files. There are two other pieces of information that are not mandatory, but will help in the tracking of the different report versions. The name of the person requesting the information and the name of the report are non-mandatory.

The input file is an ASCII file that contains the names Ada source file (s) to be processed by this tool. Even though there exist a cost in time to generate the file with all the

filenames, the benefits of this approach outweigh the cost. (See Figure 7) The output file

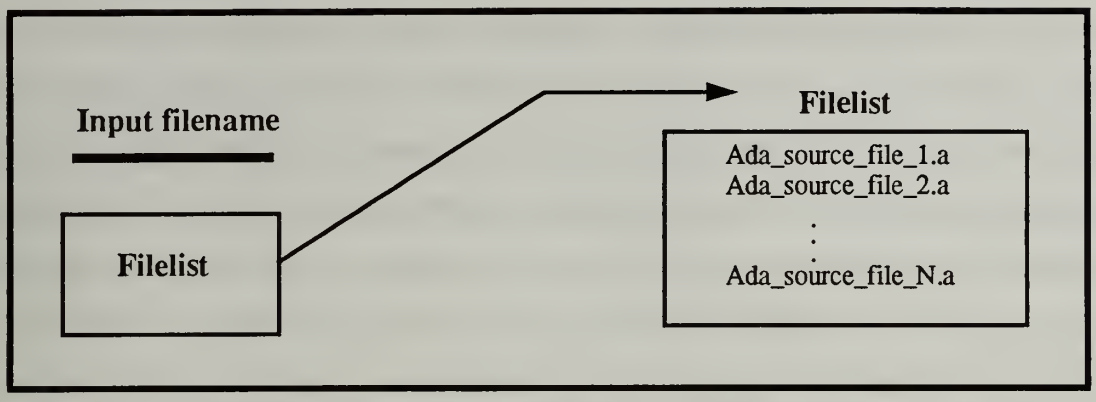

Figure 7 Relationship of input file name and Ada source files to be measured

name is used to create an ASCII file containing the reports requested by the user. At this time there ia no default name used by the tool.

2. Size Attributes

The five attributes of the SEI framework that are measured in this tool have each been declared as enumerated types. The values for each attribute enumerated type are the values for that attribute as defined in the SEI framework for size checklist. Using enumerated types facilitated the use of the Ada language attributes associated with enumerated types. (See Figure 8) .

```
type STMT_TYPE is (EXECUTABLE, DECLARATIONS ,
                   COMPILER_DIRECTIVES,
                   CMTS_ON_OWN_LINE,
                   CMTS_WITH_SRC_CODE,
                   BANNERS_NON_BLANK_SPACERS,
                   BLANK_COMMENTS, BLANK_LINES);
```

Figure 8 Attribute Statement Type values declared as an enumerated type

3. Five Dimensional Arrays

To keep track of the five orthogonal attributes a five dimensional array was created, one dimension corresponding to each attribute. This was followed by the creation of six instances of the five dimensional array. Each instance of the five dimensional array is associated to one of the six reports. Each line counted of the Ada source files has one of the values of each attribute associated with it. The Automated Ada Physical Source Line Counter uses the five-dimensional array to track this association. To calculate the individual totals for each value, the five-dimensional array is traversed one dimension at a time. The tool also uses the five-dimensional array when computing the requested two and three dimensional arrays for reports A through E. (See Figure 9)

```
type COUNT_ARRAY_TYPE is array (STMT_TYPE,
                                HOW_PRODUCED,
                                ORGIN,
                                USAGE,
                                DEVELOPMENT_STATUS) of natural;
```

Figure 9 Declaration of five dimensional array

4. Priority Arrays

The priority arrays are checked each time the lexical analyzer recognizes the end of the line marker. At this time, the highest priority of the statement type values found on the line is determined. When the statement with the highest priority is marked as "included", the line statement type attribute is set to this value.

For the five default reports A through E, the precedence for each report is the same and can not change. For the user defined report F, the precedence for each value of the statement type can be set according to the user's requirement. The user sets the precedence levels for report F inside of the user interface. (See Figure 10)

```
type PRIORITY_TYPE_ARRAY is array (1..8) of STMT_TYPE;
```

Figure 10 Declaration of Priority type array to track precedence levels

5. Flags Array

A two-dimensional array was created to determine when a particular statement type value has been recognized on a line. The first dimension of the array is the range of the enumerated statement type. The second dimension of the array contains two boolean variables. (See Figure 11) One of the boolean values is set to true when the start of a

```
type FLAGS_TYPE_ARRAY is array (STMT_TYPE, 1 .. 2) of boolean;
```

Figure 11 Declaration of FLAGS_TYPE_ARRAY

language construct is recognized by the parser. The second boolean value is set to true when the end of a language construct is recognized by the parser. (See Figure 12) When the lexical analyzer recognizes the end of line, the flags_array is traversed from highest to lowest precedence. The statement type value recognized with the highest precedence is then counted, if required.

```
set_exec_start : {FLAGS_ARRAY(STMT_TYPE'val(0), 1) := true; };
set_exec_end : {FLAGS_ARRAY(STMT_TYPE'val(0), 2) := true; };
```

Figure 12 Example of FLAGS_TYPE_ARRAY values set to true

6. Current Settings Record

In order to track each of the five dimensions, the record structure "current_settings" was created. This record structure is used whenever a physical line of code is counted. The respective current_settings field is updated when the parser recognizes either the language constructs associated with the statement type attribute or the special comments for the other four attributes.

7. Checklist Variables

A record structure, record_flags, was created to track each attribute and its values. There is one instance of record_flags for each report. The default values for reports A

through E are assigned when the package Global is elaborated. The default values for report F are the same as the basic report A. The user specifies their own values for report F by stepping through the user interface panels. See Appendix B for copy the global package source code.

F. OVERVIEW

The Automated Ada Physical Source Line Counter, as its name suggests, is a tool that will perform a count on Ada source files and generate reports of the total number of physical source lines counted and individual totals of each value included for each report requested. The tool consists of four parts diagramed in Figure 13.

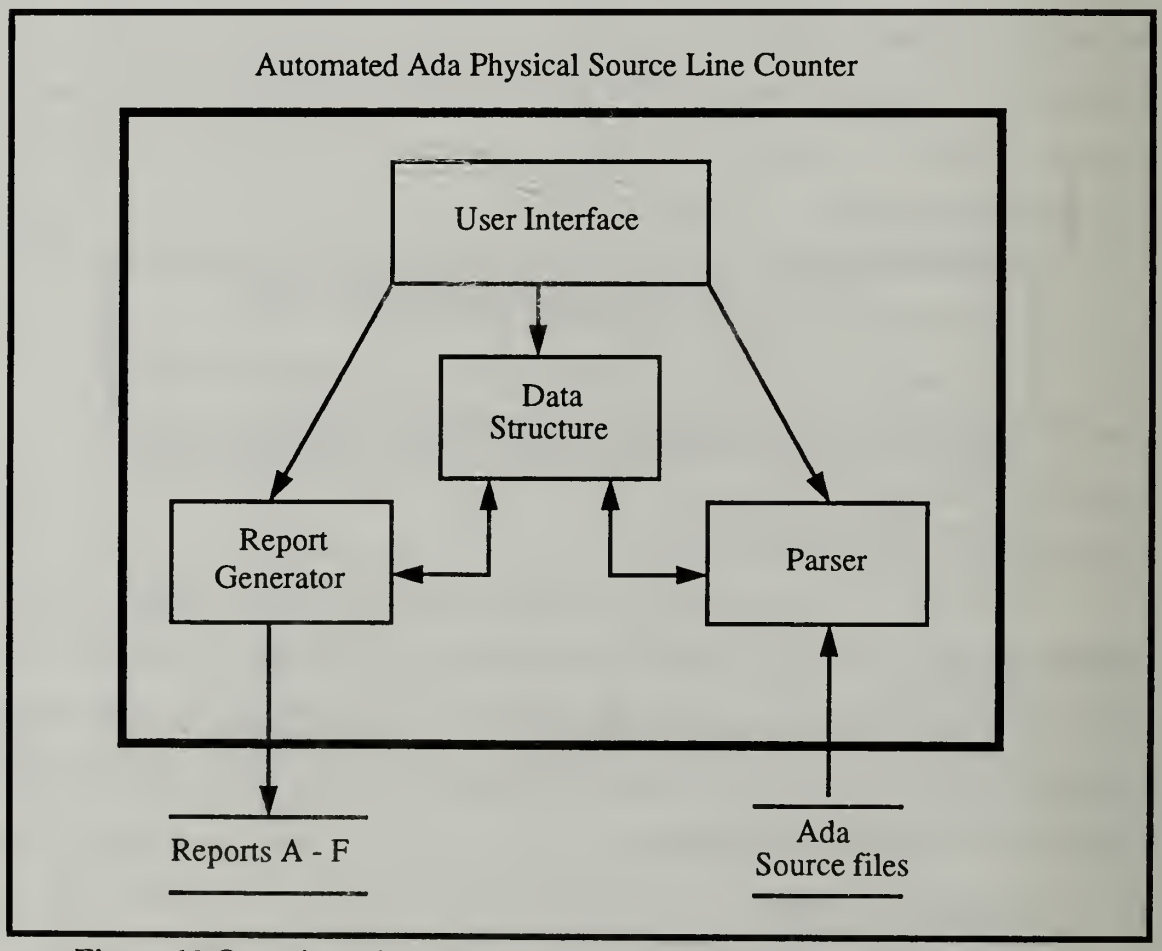

Figure 13 Overview of the Automated Ada Physical Source Line Counter

The first part is the data structure. The data structure holds all of the default and user set variables that are used by the other three parts of the tool. The default values are set for reports A through E during elaboration. The user-specified values for report F are set by the user via the user interface. The second component is the user interface. The user interface allows the user to request either five default reports, or to create a report of their own. The user interface was built using TAE, and based upon the SEI framework on size checklists.[NAS 90] The next element of the tool is the parser. The parser was created using two tools, Ayacc, and Aflex. Ayacc generates a parser, where Aflex generates a lexical analyzer used by Ayacc. The parser created by Ayacc and Aflex is used to distinguish between the different values of the statement type attribute, recognize special comments, and at the end of every line determine how the line should be counted, if at all. The next part is the report generator. This part performs the calculations that sum the individual and aggregate totals and any two and three-dimensional arrays. In addition, each report that is requested is generated and written to an ASCII file. Each part of the tool will be discussed in greater detail in the following sections. A user manual for the tool is included in Appendix A.

G. USER INTERFACE

The user interface provides a window type access that is an easy-to-use method to request one or several pre-defined reports or to create an individualized report. Each of the supported attributes is contained in its own panel or screen. The user interface is made up of eleven panels. (See Figure 14)

The panels are made up of selection items, text items and labels. There are three types of selection items. The types are push-button, checkboxes and radio buttons. The push-button is used to connect one panel to another. The push-buttons are shaped like a rectangle. The checkboxes are used whenever the user has the choice to pick more than one item. The checkboxes are shaped like a square. For example the user can pick just one report, say A, an or the user can pick all six reports, A through F. The radio buttons are used when the

user can pick only one of the items in the group. The radio-buttons are shaped like a diamond. At least one item will always be picked. For example, if the user picks the value blank lines to be included in report F, then the radio button for "Includes" will be highlighted. The button for "Excludes" will change from highlighted to blank and vice versa.

Each of the panels have default settings for push-buttons, radio-buttons and checkboxes. Each default selection is highlighted. To change or add too the default selection, the user must use the left mouse button. The default push-button can be selected when the return key is pressed while the cursor is in that panel.

The push-buttons for each panel are displayed along the bottom. Two of these push-buttons are common to each panel and will be discussed separately from each particular panel. The first push-button is the "Quit" button. The other push-button is the "Help" button. When the quit button is pressed, the quit panel is displayed over the top of the current panel. The quit panel gives the user the choice to quit the application, or to go back to the panel that they were just on. When the help button is pressed, a help panel with information particular to that panel will be displayed. When the user is finished with the help screen, the help screen will disappear and the panel that initiated the help screen will again be the active screen.

The first panel is an introduction panel. The introduction panel contains the name of the tool, name of the author and three push-buttons displayed along the bottom. Beside the quit and help push-buttons, the other push-button is the "Next screen" button. The next screen button will make the introduction panel disappear, and bring up the second panel. The next screen push-button is the default push-button for the introduction panel.

The second panel is where the user will enter the mandatory information for the tool to operate. The second panel contains four string keyin areas, a group of six checkboxes and four push-buttons. The string keyin areas are for the report name, file list, requestor name and output file name, respectively. The user enters the appropriate information by placing the cursor over the window and type in the appropriate information. Six checkboxes

represent six different reports that can be generated by this tool. Any one or all of the checkboxes may be selected. Report A is the default selection. In addition to the quit and help push-button, the second panel also has displayed along the bottom a "Generate Report" push-button and a "Specify Custom Report" push-button. When the generate report button is pressed, the second panel will disappear and the generate report panel will appear on the screen. When the specify custom report push-button is selected, the second panel disappears, and activates the third panel. The generate report push-button is the default push-button for the second panel.

The third through the seventh panels contain the attributes of the SEI checklist, one attribute per panel. There are five push-buttons on each of these panels. The first two push-buttons are the quit and help buttons. Another push-button is the "Previous Screen" push-button. When the previous screen button is pressed, the current panel disappears, and activates the previous panel. The next push-button is labeled "Next Screen". When pressed, the current panel will disappear, activating the next panel in the sequence. The last push-button is the generate report button. When this button is pressed it will make the current panel disappear and activate the generate report panel. The generate report panel is the default button.

Along the top right corner in panels three through seven are two radio-buttons displayed, one above the other. These radio-buttons allow the user to specify that in addition to the individual totals, this attribute will be included in a multi-dimension array at the end of the normal report format. When this choice is selected, all of the attributes selected as such (must have at least two) will be displayed as two or three dimensional arrays at the end of report F. When more than three attributes are selected, then all combinations of N choose three will be displayed at the end of report F, where N will be either four or five. The other major part of panels three through seven are the radio-buttons that correspond to the values for each attribute. All of the radio-buttons are the same, either the include button is highlighted or the exclude button is highlighted. Panel three also has eight integer keyin windows. Each integer keyin window corresponds to one of the values

of the attribute statement type. TAE will ensure that the precedence entered is within the range of one through eight. However, if the user does not ensure that each precedence value is unique, the results for report F may not be accurate.

The eight and ninth panels are for the general and Ada specific clarifications panels respectively. These two panels are similar to panel three, the difference being the number of radio-buttons. There are a total of thirteen general clarifications and six Ada specific clarifications. Each clarification is associated with one of the values of the attribute statement type.

The tenth panel is the generate report panel. This panel has two push-buttons displayed along the bottom of the panel, they are labeled cancel and generate report. The generate report is the default button. The cancel button will make the generate report panel disappear. The user must then use the mouse to click on the icon of the previous panel. In addition to the push-buttons, this panel displays a text message explaining the different options available to the user.

The last panel is the quit panel. This panel also has two push-buttons displayed along the bottom of the panel, they are labeled quit and cancel. The quit button is the default button. When the quit button is selected, the panel will disappear and the tool will terminate. When the cancel button is selected, the quit panel disappears leaving the previous panel as the active panel. In addition to the push-buttons, there is a text area that displays the options to the user. This is provided in lieu of a help button.

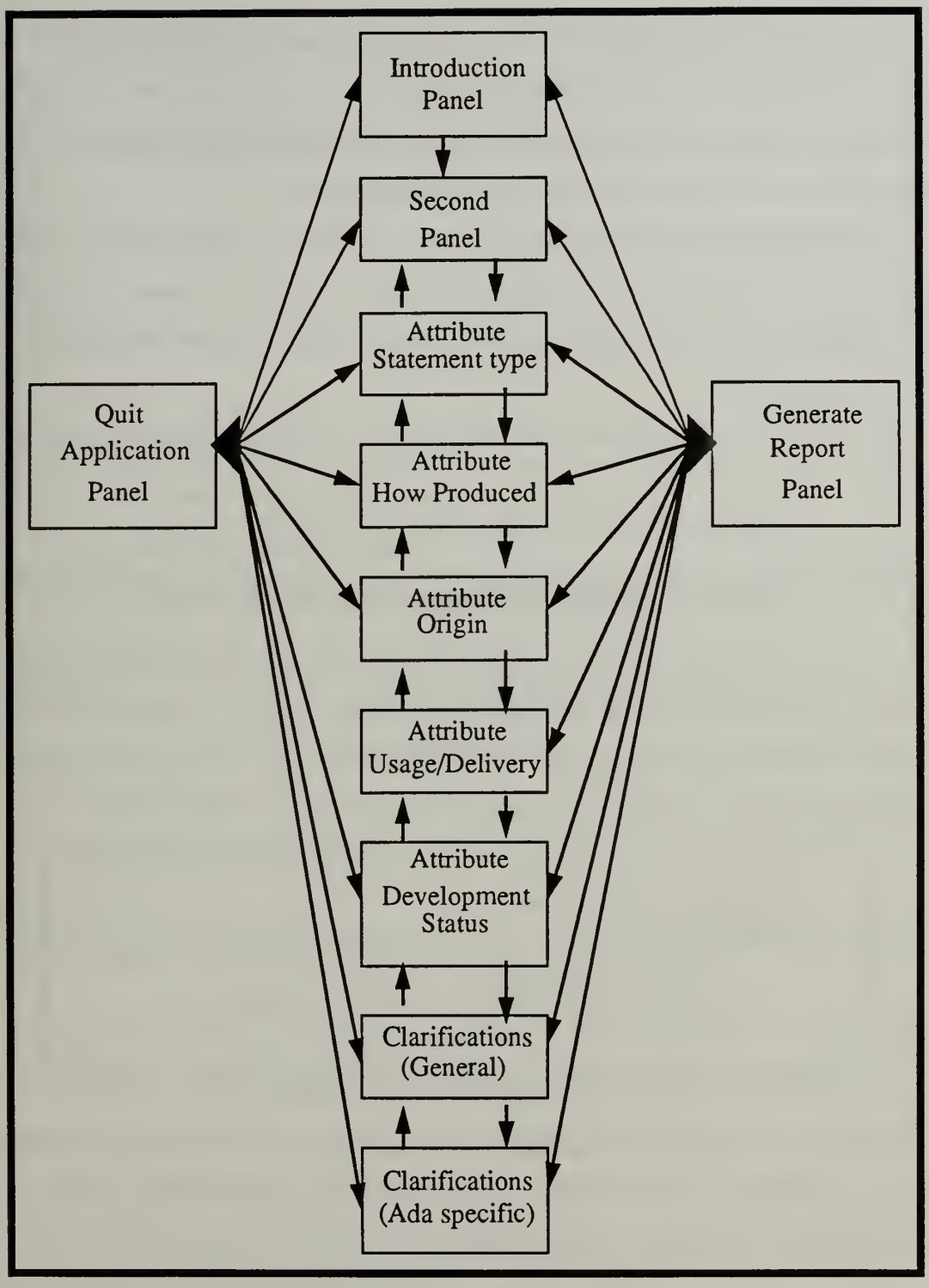

Figure 14 Overview of User interface

H. PARSER

The Automated Ada Physical Source Line Counter uses the generated parser from Ayacc [TAB 88] to differentiate between executable, declarations and compiler directives. These are three of the eight values associated with the attribute statement type. To do this, the specification file for Ayacc, ada.y, was slightly modified.

The differences between the three attributes were used to set flags when a particular language construct was recognized. To set these flags, several nonterminals were added to ada.y. The purpose of these nonterminals was to have the parser execute the associated Ada code. (See Figure 15)

```
set_exec_start : {FLAGS_ARRAY(STMT_TYPE'val(0), 1) := true; };

set_exec_end : {FLAGS_ARRAY(STMT_TYPE'val(0), 2) := true; };
```

Figure 15 Example of Ada code executed inside of Ayacc

The Automated Ada Physical Source Line Counter uses the lexical analyzer generated by Aflex [SEL 90] for several purposes. The primary reason is to provide the lexical analyzer function required by Ayacc. This tool also uses Aflex to find all occurrences of each type of comment and blank lines in the Ada source files. (See Figure 16)

```
-- Checking for blank lines
^[\t]*\n {ECHO;
        FLAGS_ARRAY (STMT_TYPE'VAL (7), 1) := TRUE;
        ADD_TO_ARRAY;
        linenum;}
```

Figure 16 Example of rule to find blank lines

This tool also uses the lexical analyzer to recognize certain Ada source statements/ fragments in certain situations. These situations have been derived from the clarifications code (general and Ada specific). For example, rules were added to the Aflex specifications file which will find when one of the following occurs: an "elsif" on line by itself; an "else" on line by itself; a "then" on line by itself; or "others" on line by itself. (See Figure 17)

```
---- Looking for an elsif on a line by itself
-- ^[\t]*"elsif"[\t]*\n {ECHO; ENTER(Z);
if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_11) then
                                FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
                                FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
                                ADD_TO_ARRAY;
end if;
 linenum;
 return(ELSIF_TOKEN);}
-- Looking for an "else" on a line by itself
^[\t]*"else"[\t]*\n {ECHO; ENTER(Z);
if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_10) then
                                FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
                                FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
                                ADD_TO_ARRAY;
 end if;
return(ELSE_TOKEN);}
 linenum;
```

Figure 17 Example of rules added to Aflex

Finally, this tool uses the lexical analyzer from Aflex to recognize when any of the 27 special comments or flags have been used in the Ada source files. The special comments are used to change the values of the four attributes How Produced, Origin, Usage, and Development Status. When a special comment is found, Ada code is executed to set the global flag SPECIAL_COMMENT to true and to set the associated current_settings field to the corresponding value. See Figure 18.

```
"--*_Generated" {ECHO;
                 SPECIAL_COMMENT := TRUE;
                 CURRENT_SETTINGS.SECOND_ATTRIBUTE := HOW_PRODUCED'val (1);}
```

Figure 18 Example of code to recognize Special comments

When the lexical analyzer reaches the end of each line, the procedure ADD_TO_ARRAY is called. This procedure determines which reports are active, determines the highest priority of the statement type(s) recognized on the line, and determines if this statement type value is included or excluded for the reports that are active. If the statement type value is included, then the count for that report is incremented.

I. REPORT GENERATOR

The report generator provides for the generation of the reports after the Ada source files have been parsed. The report generator is made up of several Ada packages. Two of which will be discussed here. The main package is the report_package. The main functions of this package are to determine which reports have been requested; perform the necessary calculations of the values for each report; and to create and write to the output file the reports requested. Several of the default reports require the generation and output of two or three dimensional arrays. To accomplish this, a separate generic package was created.

All calculations of the two and three dimensional arrays are performed in the generic package. The large number of multi-dimensional arrays that could occur in report F was the driving factor for this package. In report F, the user can request data arrays for all five attributes. This would require ten two-dimensional arrays of five choose two and ten three-dimensional arrays of five choose three. To reduce the number of instantiations of the generic package, the permutations of the two and three dimensional arrays were checked and the duplicates were discarded. The generic package is instantiated fourteen times.

J. SUMMARY

This chapter discusses the attributes supported, attributes not supported, default reports A through E, user defined report F, key data structures, user interface, parser and

the report generator. Together, these parts show that the SEI framework for Size can be implemented into a fool providing flexibility and maintaining the two criteria of communication and repeatability.

IV. TOOL USAGE

A. INTRODUCTION

This chapter describes what is needed to use the tool in section B. Some limitations of the tool are discussed in section C. Section D goes over the command line invocation. An extended example of an Ada source file is discussed in Section E. Finally, section F discusses the reports generated as output from the extended example. Appendix A provides a complete user manual for the tool.

B. REQUIREMENTS

1. Hardware

The Automated Ada Physical Source Line Counter requires the use of an Unix workstation. The tool has successfully run on several SPARC compliant computers: Solbourne Computer S4000, Sun SPARC station 10, Sun SPARC station 1 and Sun SPARC station 2.

2. Software

The Automated Ada Physical Source Line Counter requires the use of "X-windows" to operate. This tool has worked under Openwindows and Motif.

3. Input

The Automated Ada Physical Source Line Counter requires the entering of several pieces of information to run correctly. The information that is required are the input filename, output filename, name of person requesting report and name of the report. The first two items are required information. The second two pieces of information are not required, but suggested.

4. Legal Ada Syntax

The Automated Ada Physical Source Line Counter will only work with syntactically correct Ada source files. In some instances, generated code will have

embedded special characters, such as (^L, page breaks for printing) that will cause a syntax error in the parser. This particular error does not cause the tool to terminate, but there may be some embedded characters that do. This prototype tool was built using version 1.0 of Ayacc [TAB 88]. A newer and improved version of Ayacc was released after the tool was built and offers some improvement in acceptance of Ada source files.

C. LIMITATIONS

1. Package Conflicts

The parser generated by the two tools Ayacc uses a grammar supplied be the user. [TAB 88] For this tool, the grammar was the one supplied with Ayacc, but modified for purposes of the tool. For the situation of recognizing either a package spec, a package body or a generic package requires that for the proper counting of lines, the entire package spec, package declaration or generic package declaration must be one line or the final count may be incorrect.

```
gen_inst :
        PACKAGE_TOKEN IDENTIFIER IS_TOKEN
        NEW_TOKEN expanded_n .gen_act_part. ';'
    |   PROCEDURE__ident__IS_
        NEW_TOKEN expanded_n .gen_act_part. ';'
    |   FUNCTION_TOKEN designator IS_TOKEN
        NEW_TOKEN expanded_n .gen_act_part. ';' ;
```

Figure 19 Example of Ada.y input file to Ayacc

2. Coding Style

The Automated Ada Physical Source Line Counter counts physical source lines of code and is based upon the SEI framework on size. [SEI-B 92] Different coding styles can and will result in different results. For example, let's compare a short example of the same code, but different writing styles. (See Figure 20) The total number of non-comment, non-blank lines for version one would be two. However, the total for non-comment, non-

blank lines for the same exact code in version two results in a total of five. The use of a pretty printer on the Ada source files prior to using the Automated Ada Physical Source Line Counter will ensure consistent results for reports A through E.

Version One

```
procedure EXAMPLE_ONE is TEMP_INT : integer:= 0; begin
TEMP_INT := TEMP_INT + 1; end EXAMPLE_ONE;
```

Version Two

```
procedure EXAMPLE_TWO is
   TEMP_INT : integer:= 0;
begin
   TEMP_INT := TEMP_INT + 1;
end EXAMPLE_TWO;
```

Figure 20 Example of two different coding styles

D. COMMAND LINE INVOCATION

To start the Automated Ada Physical Source Line Counter, the user must either be in the directory that contains the tool or have the directory containing the Automated Ada Physical Source Line Counter in a valid path statement. In addition, the user must be running in an X-windows environment (operating motif, for example). The tool is started by any Unix program-execution procedure, but input and output may not be redirected.

E. EXTENDED EXAMPLE

1. Sample Application

The package TASK_PACKAGE was created as part of an earlier class programming project. This package just one of several packages created for a class project. The overall objective of this project was to read in a file containing initial information about

a spaceship. The information included location in three-dimensional space, speed in each of the three directions and remaining fuel. The object of the program was to be able to accept input from a user via the keyboard at least every second and update the spaceships parameters. This file was chosen for the extended example because it was of moderate length, involved a number of different Ada statements in one file.

2. User Interface

In order to see the actual measurement results for an Ada source file performed by this tool, the tool was invoked with the file EXAMPLE_FILE. This file contained the Ada package TASK_PACKAGE. (See Figure 21) This section works through an example

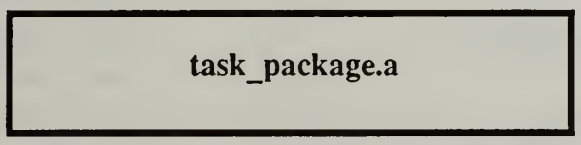

Figure 21 Contents of EXAMPLE_FILE

of how to use the tool and how the tool works. This example will result in the generation of reports A through F. The tool is started as explained in section A. The user then places the cursor inside of the introduction panel and presses the carriage return. The information needed to properly run the tool is typed into the text keyin fields of panel number two. (See Figure 22) The tool needs this information in order to know the correct input and output files to open and create respectively. In addition, the name of the report and person requesting report are entered. The next step is to choose the report(s) the tool are to generate. (See Figure 23) Report A is already marked so no further action is required. For reports B through F the corresponding checkbox must be marked by placing the cursor over the checkbox and/or label for reports B through F and press the left mouse button. Since report F has been chosen, the next step is to place the cursor over the push-button Specify Custom report. This action will cause panel number two to disappear and activate panel number three.

The changes for panel three include marking all of the radio-buttons for the attribute statement type values as included and selecting the push-button generate report.

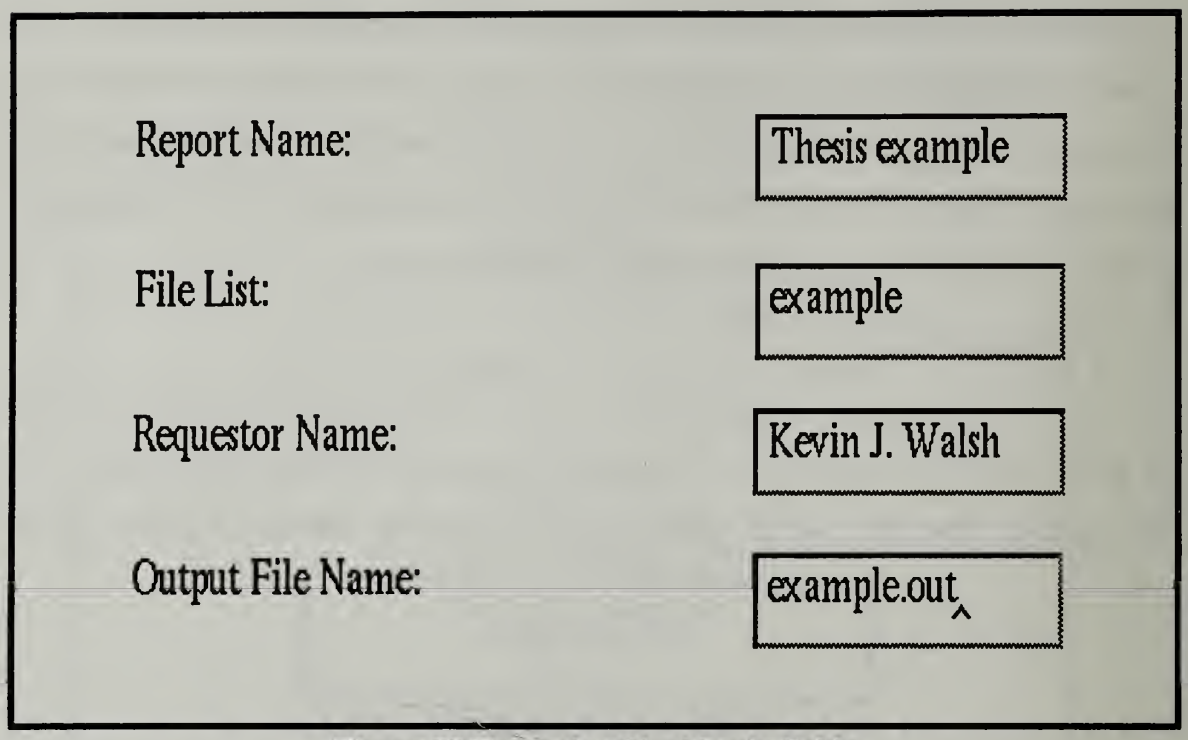

Figure 22 Example of text keyin fields

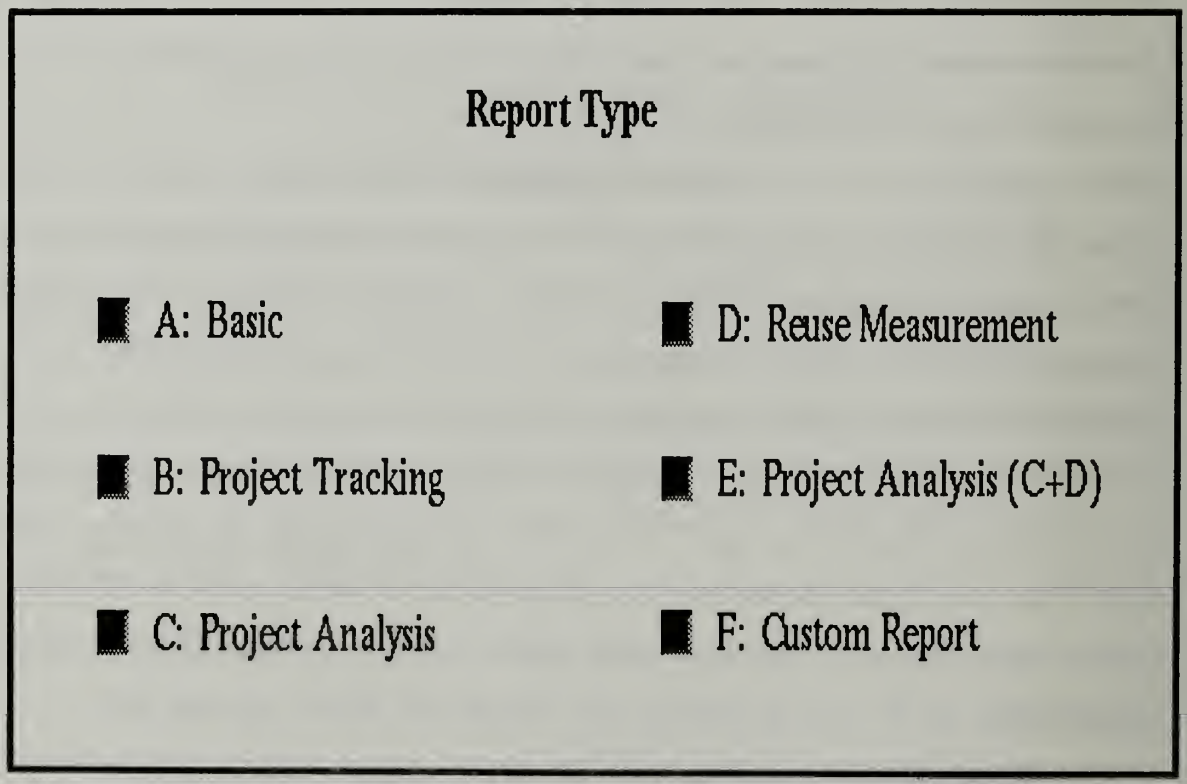

Figure 23 Example of marked checkboxs for selecting reports A - F

(See Figure 24) Marking all of the radio-buttons as included customizes report F to measure all lines of the input Ada source file as one of the eight possible statement type values. When the push-button generate report is selected, panel number three will disappear and the generate report panel appears. Since no mistakes have been made, the next step is to select the push-button generate report. The generate report panel will disappear.

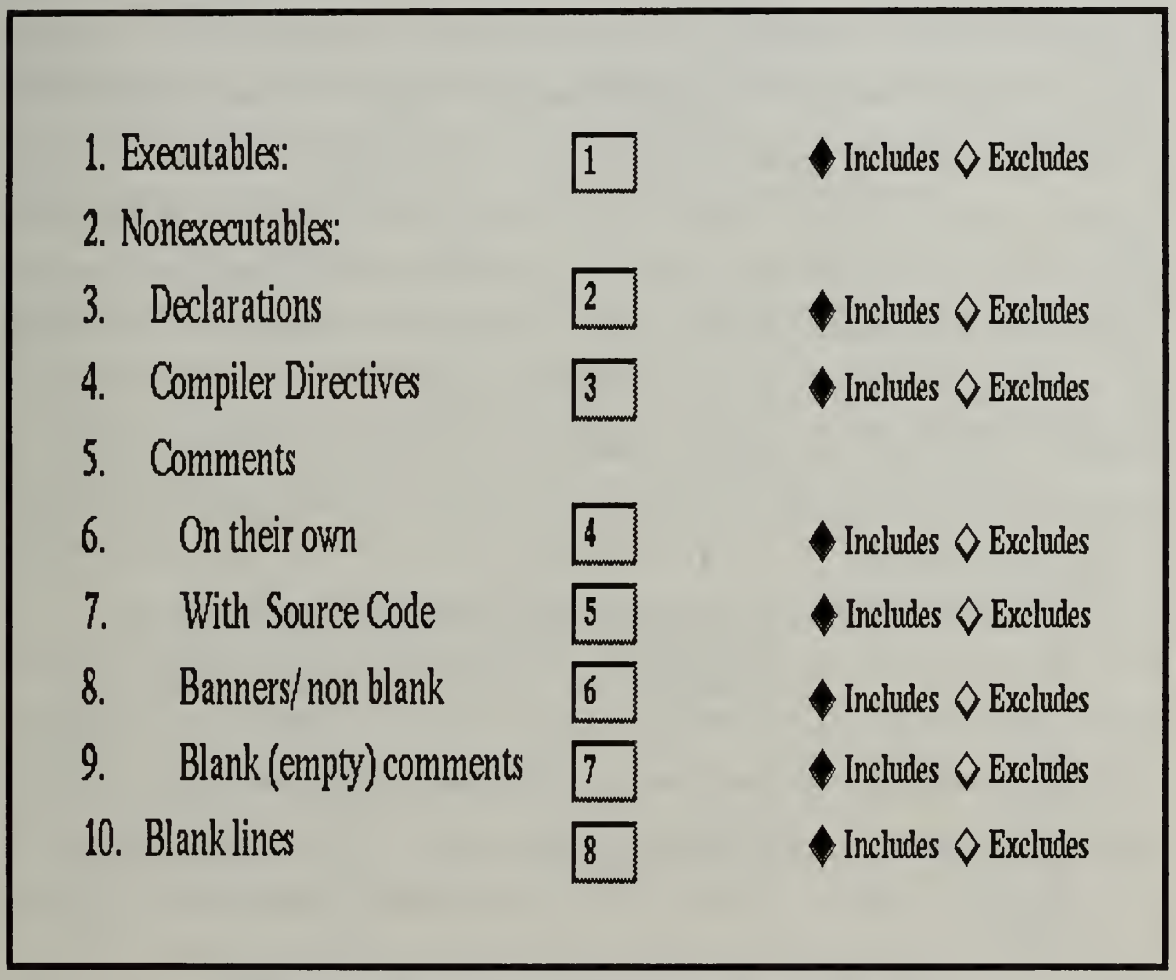

Figure 24 Example of customizing report F

The user interface portion of the tool is now complete. Control of the tool now passes over to the parser section. The file list file name entered in panel two is opened to read the name(s) of the Ada source file(s). In this example, the file task_package.a is opened and the parser starts to work. To explain how the parser section of the tool works,

an example of five of the possible values of the attribute 'statement_type' are discussed in the following paragraphs.

3. Statement Processing

The flexibility of the Ada programing language made it necessary to create several variables that this tool and in particular the parser uses to recognize the various values of the attribute 'statement_type'. The Ada language allows more than one statement type on a line and allows for executable statements, declarations and compiler directives to extend over more than one line. To account for this, the parser needs to mark the start and end of executable statements, declarations and compiler directives; and also to track when either an executable statement or declaration or compiler directive extends beyond one line. The variables are outlined by statement type and function in the figure below. (See Figure 25) The variables used for compiler directives is included in Figure 25, even though the extended example does not include an example of a compiler directives, the function and actions are similar to executable and declaration statements. The code used for the extended example is in Figure 26.

	Mark start of statement variable	Mark end of statement variable	Track multi-line statement variable
Executable	exec_start	exec_end	exec_level
Declaration	dec_start	dec_end	dec_level
Compiler Direc-tives	pragma_start	pragma_end	pragma_level

Figure 25 Variables used during parsing of source files

a. Executable Statements

The parser processes the Ada source sequentially, reading in tokens via the lexical analyzer. When the parser recognizes the keyword 'select' as starting an executable statement, exec_start is set to true (See bubble 1 in Figure 26). When the lexical analyzer reaches the end-of-line marker and the procedure add_to_array is called.

Once inside add_to_array, the values of exec_start and exec_end are evaluated to determine if the variable exec_level needs to be increased or decreased. In this case, the variable exec_level is increased. The next check made is to determine if this line of code is either a comment on its own line, a banner comment, an empty comment or a blank line. For this line, all of these cases fail.

The next part of the code checks from the highest to the lowest precedence statement type. In this case, the execution statement precedence is the highest, so the loop is traversed only once. Since the exec_start flag is true, the current_settings.fist_attribute is set to executable. Now the line is ready to be added to all applicable arrays.

The procedure determine_which_array is called with the variable current_settings. This procedure will add this line to all applicable arrays as long as all five values of the variable current_settings are valid for each report requested. In this case, all six reports have been requested and the values for the variable current_settings are valid for each report. The corresponding entry in the arrays for each report is increased by one. Once the arrays are modified control is passed back to the parser.

For the second through the fourth line of this executable code, the variable exec_level is greater than zero and the variables exec_start and exec_end are false. The lexical analyzer reaches the end-of-line marker and checks the status of the flag settings. Since the variable exec_level is greater than zero, then the variable exec_start is set to true. In this situation, the variable current_settings.first_attribute is again set to executable, and then processed as the first statement.

At the beginning of the last executable line is this example, exec_level is greater then zero. When the parser reaches the 'end_select', the end of the executable statement recognized and the flag exec_end is set to true. Before the flags_array is checked, the variable exec_start is set to true because of the fact that exec_level was greater than zero. Now when the flags_array is checked, both of the variables exec_start and exec_end are true. The variable current_settings.first_attribute is assigned the value of executable, and processed as were the preceding executable statements.

```
-------------------------------------------------------------
-- Task allows the user to input data to the program.
-- Task will verify input to ensure that input is valid
-------------------------------------------------------------
task body KEYREAD is

   CHARACTER_INPUT :  ROCKET_CONTROL_INPUT;
   CHARACTER_IO    :  character;
   DONE            :  boolean := FALSE;
   TEST            :  natural;

begin


   select
     accept start;
   or
     terminate;
   end select;
```

Figure 26 Sample input code

For the last executable line in this example, the variable exec_level and exec_end are both set to true. When the lexical analyzer reaches the end of the line marker, the flags_array is checked. With the executable statements having the highest precedence, they are checked first. Both of the flags are set to true, which means that at least one executable statement either started and finished on this line or an executable statement was finished on this line. In this case, it was the later of the two. The variable current_settings.first attribute is assigned the value of executable. Then the variable current_settings is checked for each report. In this case, for all reports, the current_settings are valid and each report count_array is incremented by one.

b. Declaration Statements

When the parser recognizes the identifier 'character_input' as the start of a declaration statement, the variable dec_start is set to true (See bubble 2 of Figure 26). When the parser reaches the semicolon, it recognizes the end of the declaration statement and sets the variable dec_end to true. In this case, there are no further statement types and the end-

of-line marker is reached in the lexical analyzer. At this point the procedure add_to_array is called and is processed as discussed above. In this example, all four declaration statements are parsed and counted in the same manner as the first statement.

c. Comments on Own Line

Unlike executable statements, declarations and compiler directives, all comments and blank lines are handled directly from the lexical analyzer. When the lexical analyzer recognizes a comment on a line by itself, (See bubble 3 in Figure 26), a comment flag is set to true, the procedure determine_type_comment is called first and then the procedure add_to_array is called. The lexical analyzer recognizes a comment on a line by itself using the following rules:

- Zero or more spaces or tabs between the start of line marker and two hyphens.
- Any combination of one or more characters between the two hyphens and the end of line marker.

The procedure determine_type_comment is passed the length of the current line, the third character the line and the string of characters from one to the current line length. The main purpose of determine_type_comment is to see if the current comment being parsed is either a regular comment on a line by itself or a banner comment. The actions taken for banner comments are discussed below. In this case, the comment is not a banner comment.

Once inside add_to_array, the first check that is applicable for comments is determining if the current comment is a full line of code. This is true for this case. The current_settings.first_attribute will be set to comments_on_own_line and the procedure determine_which_array will be called. As discussed earlier, the procedure determine_which_array will check all of the five fields of the variable current_settings and increase each array by one when no discrepancies are found. In this case, only the arrays for reports C, E and F are incremented. They are the only reports that have comments on own line marked as included.

d. Banner Comments

A banner comment is a line of symbols used to visually separate blocks of comments or blocks of source code. (See bubble 4 in Figure 26) The actions taken for banner comments are similar to regular comments except as noted here. Inside of the lexical analyzer there are two places where a banner comment may be recognized. The first rule is the same rule discussed above for comments. The second rule that looks for a banner comment made up of just hyphens. The rule used is as follows:

- Between the start of line marker and the first three hyphens, there can only be zero or more blanks and or tabs.
- Following the first three hyphens there can be zero or more hyphens.
- Between the hyphens and the end of line marker there can only be zero or more blanks and or tabs.

In the first case the procedure determine_type_comment is called. The comment is then parsed looking for a repetition of the third through the sixth character. The third through sixth character needs to repeated at least four times to count the comment as a banner comment. If the comment meets the criteria, then the start and stop flags for banner comments are set to true. If the comment does not meet the criteria then the start and stop flags for a comment on a line by itself are set to true. The procedure add_to_array is called next. The actions taken are similar to those discussed above.

e. Blank Lines

When the lexical analyzer recognizes a blank line (see bubble 5 in Figure 26) the blank_line flag is set to true and the procedure add_to_array is called. In this example, for any of the blank lines, the start and end flags for blank lines will be set to true.

Once inside add_to_array, the first check that is applicable for blank lines is determining if line of code in question is a full line of code. This is true for this example. The current_settings.first_attribute will be set to blank_lines and the procedure determine_which_array will be called. As discussed earlier, the procedure determine_which_array will check all of the five fields of the variable current_settings and

increase the corresponding entry of each array by one if no discrepancies are found. In this case, this line will only be added to report F. Reports A through E do not have blank lines marked as included, therefore their arrays are not increased.

f. OUTPUT

After the source files have been parsed and measurements collected, the data must be presented in a way that can be read and understood by the people who request the measurements. For the example of using task_packag.a as the Ada source files, a copy of each possible report was requested and generated. The different reports will show how using different rules can result in different but correct results.

For brevity, only parts of each report are shown. The complete listing of each report are included in Appendix C. The file task_package.a was processed by the Automated Ada Physical Source Line Counter producing reports A through F. (See Figure 27) (See Figure 28) (See Figure 29) (See Figure 30) (See Figure 31) (See Figure 32) To compare the results of this tool, the Unix utility wc [SUN 90] was also ran on the file task_package.a, which calculated a total of 284 lines for the file. Reports A through the F also report the number of lines, but also provide additional information as detailed in the SEI Framework for Size Measurement. [SEI-B 92]

Report A is the tools basic definition for counting physical source lines of code. Report A details the total number of lines and individual totals for each value marked as included. Report A measures all noncomment and nonblank physical source line. (See Figure 27) Report A measured a total of 193 lines of physical source lines of code.

Report B is an example of a report that can be used for project tracking. The results from this report can be used to track development status. Report B measures the total number of lines, individual totals for values marked as included and a two-dimensional array consisting of the attributes development status and how produced. (See Figure 28) This report will also count any removed code if annotated with a special comment. Report B measured a total of 193 physical source lines of code.

REPORT A

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 193
Estimated: 0

		Total Includes	Total Excludes	Individual totals
Statement type				
When a line or statement contains more than one type, classify it as the type with the highest precedence.				
1 Executables	Precedence => 1	XXXX		157
2 Nonexecutables				
3 Declarations	2	XXXX		36
4 Compiler Directives	3	XXXX		0
5 Comments				
6 On their own lines	4		XXXX	0
7 On lines with source code	5		XXXX	0
8 Banners and nonblank spacers	6		XXXX	0
9 Blank (empty) comments	7		XXXX	0
10 Blank lines	8		XXXX	0

Figure 27 Partial output of Report A

Report C is an example of a report that can be used for project analysis. This report would usually be requested only at the end of a project. The results would be used to provide for better estimates of future projects. Report C measures the total number of lines, individual totals for all values marked as included and a two-dimensional array consisting

REPORT B

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 193
Estimated: 0

		Total Includes	Total Excludes	Individual totals
Statement type When a line or statement contains more than one type, classify it as the type with the highest precedence.				
1 Executables	Precedence => 1	XXXX		157
2 Nonexecutables				
3 Declarations	2	XXXX		36
4 Compiler Directives	3	XXXX		0
5 Comments				
6 On their own lines	4		XXXX	0
7 On lines with source code	5		XXXX	0
8 Banners and nonblank spacers	6		XXXX	0
9 Blank (empty) comments	7		XXXX	0
10 Blank lines	8		XXXX	0

Figure 28 Partial output of Report B

of the attributes statement type and how produced. Report C measured a total of 240 physical source lines.

Report D is an example of a report used for reuse measurement. The results of this report can be used to evaluate the amount of software reuse. Report D measures the total number of lines of lines, individual totals for all values marked as included and a two-

REPORT C

Report Name: Thesis example

File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 240
Estimated: 0

		Total Includes	Total Excludes	Individual totals
Statement type When a line or statement contains more than one type, classify it as the type with the highest precedence.				
1 Executables Precedence =>	1	XXXX		157
2 Nonexecutables				
3 Declarations	2	XXXX		36
4 Compiler Directives	3	XXXX		0
5 Comments				
6 On their own lines	4	XXXX		47
7 On lines with source code	5	XXXX		0
8 Banners and nonblank spacers	6		XXXX	0
9 Blank (empty) comments	7		XXXX	0
10 Blank lines	8		XXXX	0

Figure 29 Partial output of Report C

dimensional array consisting of the attributes how produced and origin. For this example file, report D measured a total of 193 lines of physical source lines of code.

Report E is an example of the combination of two previous report specifications. Report E is also used for project analysis. This report would be requested at the end of a project. The results would then be used to better estimates for future projects.

REPORT D

Report Name: Thesis example

File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 193
Estimated: 0

		Total Includes	Total Excludes	Individual totals
Statement type When a line or statement contains more than one type, classify it as the type with the highest precedence.				
1 Executables	Precedence => 1	XXXX		157
2 Nonexecutables				
3 Declarations	2	XXXX		36
4 Compiler Directives	3	XXXX		0
5 Comments				
6 On their own lines	4		XXXX	0
7 On lines with source code	5		XXXX	0
8 Banners and nonblank spacers	6		XXXX	0
9 Blank (empty) comments	7		XXXX	0
10 Blank lines	8		XXXX	0

Figure 30 Partial output of Report D

Report E measures the total number of lines, individual totals for all values marked as true and a three-dimensional array consisting of the three attributes how produced, statement type and origin. For this example file, report E measured a total of 240 physical source lines of code.

REPORT E

Report Name: Thesis example

File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 240
Estimated: 0

		Total Includes	Total Excludes	Individual totals
Statement type When a line or statement contains more than one type, classify it as the type with the highest precedence.				
1 Executables	Precedence => 1	XXXX		157
2 Nonexecutables				
3 Declarations	2	XXXX		36
4 Compiler Directives	3	XXXX		0
5 Comments				
6 On their own lines	4	XXXX		47
7 On lines with source code	5	XXXX		0
8 Banners and nonblank spacers	6		XXXX	0
9 Blank (empty) comments	7		XXXX	0
10 Blank lines	8		XXXX	0

Figure 31 Partial output of Report E

Report F is a user_defined report. The user can change any of the values for each of the five attributes supported from included to excluded or vice versa. In addition, the user can request any combination of two, three, four and five dimensional arrays.

However, any combination of four or five dimensional arrays (all five attributes) will be reported as ten three dimensional reports. In this example report F measures the total lines, individual totals for all values marked as included. In contrast to report A that measures only noncomment and nonblank lines, report F measures all physical source lines of code. For this example, report F measures a total of 284 physical lines of code. This is the same result as the Unix wc utility. However, report F gives the reader more information than just the total number of lines.

F. SUMMARY

In summary, this chapter has discussed the tool requirements, tool limitations, command line invocation, an extended example and reports generated from the extended example.

REPORT F

Report Name: Thesis example

File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 284
Estimated: 0

	Total Includes	Total Excludes	Individual totals
Statement type When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1 Executables Precedence => 1	XXXX		157
2 Nonexecutables			
3 Declarations	2 XXXX		36
4 Compiler Directives	3 XXXX		0
5 Comments			
6 On their own lines	4 XXXX		47
7 On lines with source code	5 XXXX		0
8 Banners and nonblank spacers	6 XXXX		5
9 Blank (empty) comments	7 XXXX		0
10 Blank lines	8 XXXX		39

Figure 32 Partial output of Report F

V. SUMMARY AND CONCLUSIONS

There were two areas addressed by this research, the first was to look at the possibility of automating the SEI Framework for Size Measurement. The second was to look at how to provide the flexibility outlined in the SEI Framework for Size Measurement. This chapter provides the answers to these questions. Section A summarizes the significant results of this research. Section B concludes by giving suggestions for future research.

A. RESEARCH SUMMARY

The research studied the SEI Framework and developed a tool to implement attributes of the framework. It was determined that a prototype tool could be implemented supporting the following attributes statement type; how produced; origin; usage and development status. The user-interface was designed to mimic the SEI Checklist for each supported attribute. The user-interface calls a parser that performs the measurements according to user-defined requests. Once the parser calculates the counts for the source, the final step is to generate the user-requested reports.

After performing the development, testing and evaluation of the various features of this project, we have reached the following conclusions:

The first result of this research demonstrated that the SEI Framework for Size Measurements can be implemented in a tool using the Ada programing language. The tool consists of programed Ada code and generated Ada code. The generated Ada code was produced using TAE, Aflex and Ayacc. TAE provides the user-interface; Aflex produces a lexical analyzer; and Ayacc produces a parser. The programmed Ada code was used to integrate the three tools and to produce a report generation capability. The result is a tool that implements a major portion of the SEI Framework in Ada, with minimal execution cost.

The second outcome of this study demonstrated that the framework's flexibility can be maintained and implemented using source flags. The tool uses those flags to capture the

flexibility of SEI's framework checklist and generate multiple reports during one pass of the Ada source files. Another mechanism employed the use of global variables that are used during the parsing of the Ada code. These global variables are declared for each report to separate the different values the supported attributes may have.

B. RECOMMENDATIONS

Program managers and software developers should use tools such as the one developed in this research to track the entire software project process and compare the current program state against the estimated or planned program state at predetermined points in time. The results of the tool provide clear and consistent measurement results thereby allowing for more accurate decision making.

Since this work is among the first to use the SEI's framework, there are a large number of areas where it can be expanded with future studies. Some of these include:

The entire set of attributes outlined in the SEI Framework for Size Measurement is not implemented in this tool. The functionality and replication attributes are not supported in this tool. The attribute functionality identifies the number of lines of code that are a functional part of the code and the number of lines of code that are not functional part of the code. Whereas the attribute replication describes how to account for a software project's master source statements from its copies. To implement both of these attributes would require research into how to integrate existing tools, such as the Unix diff, or to build other tools.

The Automated Ada Physical Source Line Counter only measures physical source lines of code. The SEI framework also allows for measurement of logical source statements. To implement logical source measures research is needed to define exact and complete rules for identifying the beginnings and endings for all possible statement types.

The current tool is implemented for the Ada language. The tool can be extended to support other programming languages, such as C and C++. Development of an appropriate C parser would be needed along with interaction into the tool itself and its user-interface.

Currently the SEI framework does not involve measurements in areas outside of size, effort and quality. Research extending the SEI framework to other metric principles such as complexity is needed.

Finally, this tool provides a clear and consistent size measurement of Ada source files. The result of this and future research will improve the ability of software developers to accurately quantify and measure software projects. The metrics produced by these efforts will improve software productivity and quality. These metrics will provide additional tools to the software developer to ensure that projects meet the time and cost constraints. This research has established some initial observations and steps of how to automate the SEI Framework for Size Measurement, but more questions are left unanswered.

APPENDIX A USER MANUAL

A. REQUIREMENTS

1. Hardware

The Automated Ada Physical Source Line Counter requires the use of an Unix workstation. The tool has successfully run on several SPARC compliant computers: Solbourne Computer S4000, Sun SPARC station 10, Sun SPARC station 1 and Sun SPARC station 2.

2. Software

The Automated Ada Physical Source Line Counter requires the use of "X-windows" to operate. This tool has worked under Openwindows and Motif.

3. Input

The Automated Ada Physical Source Line Counter requires the entering of several pieces of information to run correctly. The information that is required are the input filename, output filename, name of person requesting report and name of the report. The first two items are required information. The second two pieces of information are not required, but suggested.

Special comments are used to distinguish between the different values of the four attributes how produced, origin, usage and development status. (See Figure A-1) These special comments must be entered by the code maintainer manually. These special comments flags are recognized by the lexical analyzer and change the second through the fifth field of the variable current_settings. All special comments are in the form of "--*_<text>". The double hyphens identify the line as a comment. The asterisk is included for compatibility with other tools such as Adadl. [SSD 90] The text corresponds to the unique values of the four attributes how produced, origin, usage and development status. The lines that the special comments are on are not included in the measurement.

Attributes	Description
How Programmed	
--*_Programmed	Statements prepared by programmers that are not modifications of pre-existing statements
--*_Generated	Created by using tools to produce compilable statements automatically
--*_Converted	Pre-existing statements that are translated automatically or with minor human intervention
--*_Copied	Those statements taken verbatim from other sources and used as part of the master source code for the new product
--*_Modified	Modifications are adaptations made to pre-existing statements so that they can be used in a new product, build, or release
--*_Removed	All statements that are removed from prior code when that code is copied or modified for use in a new or revised product
Origin	
--*_New_work	Statements that implement new designs
--*_Previous_version	A previous version, build or release
--*_COTS	Commercial off the shelf software
--*_GFS	Government furnished software
--*_Another_product	Another product
--*_VSL_spt_library	Vendor-supplied language support library (unmodified)
--*_VS_OS_or_utility	A vendor-supplied operating system or utility (unmodified)
--*_A_modified_spt_lib	A local or modified language support library or operating system
--*_Other_comm_lib	Other commercial library
--*_Reuse_library	A reuse library (software designed for reuse)
--*_Other_Software_component	Other software component or library

Figure A-1 Special Comments

Usage	
--*_Part_of_product	All code incorporated into the primary product and all code delivered with or as part of the product that is developed and tested as if it were to operate in the primary product
--*_External_to_product	All code that is produced or delivered by the project that is not an integral part of the primary product
Development Status	
--*_Estimated_or_planned	The total number of lines estimated for a particular software module
--*_Designed	Appropriate stage of 2167 development
--*_Coded	Appropriate stage of 2167 development
--*_Unit_tests_completed	Appropriate stage of 2167 development
--*_Integrated_into_components	Appropriate stage of 2167 development
--*_Test_readiness_review_completed	Appropriate stage of 2167 development
--*_CSCI_completed	Appropriate stage of 2167 development
--*_System_tests_completed	Appropriate stage of 2167 development

Figure A-1 Special Comments

4. Legal Ada Syntax

The Automated Ada Physical Source Line Counter will only work with syntactically correct Ada source files. In some instances, generated code will have embedded special characters, such as (^L, page breaks for printing) that will cause a syntax error in the parser. This particular error does not cause the tool to terminate, but there may be some embedded characters that do. This prototype tool was built using version 1.0 of Ayacc [TAB 88]. A newer and improved version was released after the tool was built and offers some improvement in acceptance of Ada source files.

B. LIMITATIONS

1. Package Conflicts

The parser generated by the two tools Ayacc uses a grammar supplied be the user. [TAB 88] For this tool, the grammar was the one supplied with Ayacc, but modified for purposes of the tool. For the situation of recognizing either a package spec, a package body or a generic package requires that for the proper counting of lines, the entire package spec, package declaration or generic package declaration must be one line or the final count may be incorrect.

```
gen_inst :
        PACKAGE_TOKEN IDENTIFIER IS_TOKEN
        NEW_TOKEN expanded_n .gen_act_part. ';'
    |   PROCEDURE__ident__IS_
        NEW_TOKEN expanded_n .gen_act_part. ';'
    |   FUNCTION_TOKEN designator IS_TOKEN
        NEW_TOKEN expanded_n .gen_act_part. ';' ;
```

Figure A-2 Example of Ada.y input file to Ayacc

2. Coding Style

The Automated Ada Physical Source Line Counter counts physical source lines of code and is based upon the SEI framework on size. [SEI-B 92] Different coding styles can and will result in different results. For example, lets compare a short example of the same code, but different writing styles. (See Figure A-3) The total number of non-comment, non-blank lines for version one would be two. However, the total for non-comment, non-blank lines for the same exact code in version two results in a total of five. The same code, only different coding styles. The use of a pretty printer on the Ada source files prior to using the Automated Ada Physical Source Line Counter will ensure consistent results for reports A through E.

Version One

```
procedure EXAMPLE_ONE is TEMP_INT : integer:= 0; begin
TEMP_INT := TEMP_INT + 1; end EXAMPLE_ONE;
```

Version Two

```
procedure EXAMPLE_TWO is
TEMP_INT : integer:= 0;
begin
TEMP_INT := TEMP_INT + 1;
end EXAMPLE_TWO;
```

Figure A-3 Example of two different coding styles

C. COMMAND LINE INVOCATION

To start the Automated Ada Physical Source Line Counter, the user must either be in the directory that contains the tool or have the directory containing the Automated Ada Physical Source Line Counter in a valid path statement. In addition, the user must be running in an X-windows environment (operating motif for example) To invoke the tool, any Unix program execution method may be employed, but input and output may not be redirected.

D. USER INTERFACE

The user interface provides a window type access that is an easy-to-use method to request one or several pre-defined reports or to create an individualized report. Each of the supported attributes is contained in its own panel or screen. The user interface is made up of eleven panels. The panels are made up of selection items, text items, text and integer keyin items and labels. There are three types of selection items. The types are push-button, checkboxes and radio buttons.Each of the panels have default settings for push-buttons, radio-buttons, integer keyin items and checkboxes. Each default selection is highlighted.

To change or add too the default selection, the user must use the left mouse button. Several of the figures are labeled one, two or three. The items with a one label means that those items are radio buttons. Items with a label two are push-buttons. Finally, the items labeled three are keyin items.

1. Push-buttons

The push-button is used to connect one panel to another. The push-buttons are shaped like a rectangle. (See Figure A-4) The default push-button can be selected when the return key is pressed while the cursor is in that panel.The push-buttons for each panel are displayed along the bottom. Two of these push-buttons are common to each panel and will be discussed separately from each particular panel. The first push-button is the "Quit" button. The other push-button is the "Help" button. When the quit button is pressed, the quit panel is displayed over the top of the current panel. The quit panel gives the user the choice to quit the application, or to go back to the panel that they were just on. When the help button is pressed, a help panel with information particular to that panel will be displayed. When the user is finished with the help screen, the help screen will disappear and the panel that initiated the help screen will again be the active screen.

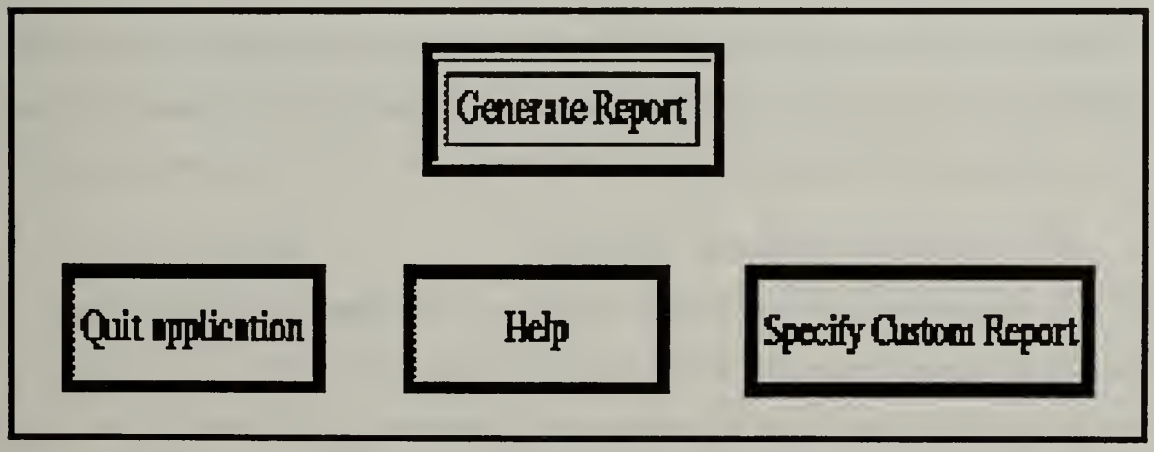

Figure A-4 Example of Push-buttons

2. Checkboxs

The checkboxes are used whenever the user has the choice to pick more than one item. The checkboxes are shaped like a square. For example the user can pick just one report, say A, an or the user can pick all six reports, A through F.

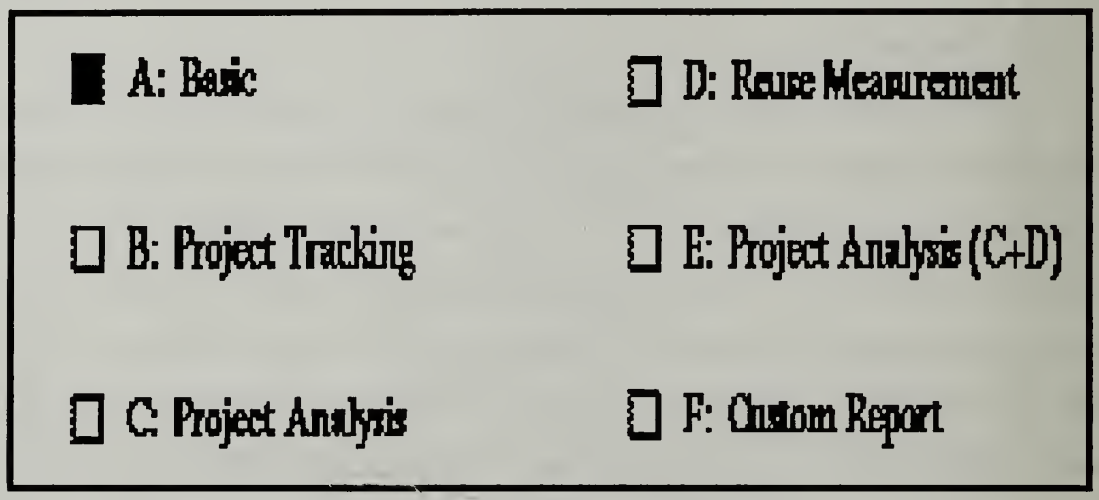

Figure A-5 Available Report Names

3. Radio-buttons

The radio buttons are used when the user can pick only one of the items in the group. The radio-buttons are shaped like a diamond. At least one item will always be picked. For example, if the user picks the value blank lines to be included in report F, then the radio button for "Includes" will be highlighted. The button for "Excludes" will change from highlighted to blank and vice versa.

4. Labels and Text/Integer Keyin items

Labels are used to identify the two types of keyin items. (See Figure A-6) The labels are place holders and have no action associated with them. The text keyin items will accept any input from the keyboard. However, if this information is the filelist, then the tool will terminate if a correct file is not found. The integer keyin fields are used for the setting of the precedence levels for report F. The precedence levels for reports A through E are preset and can only be changed by going into the source code and manually changing the

values. For the integer keyin fields, TAE will check to ensure that the value is within the prescribe range, which is from one through and including eight. However, TAE does not check to see if duplicate values are entered.

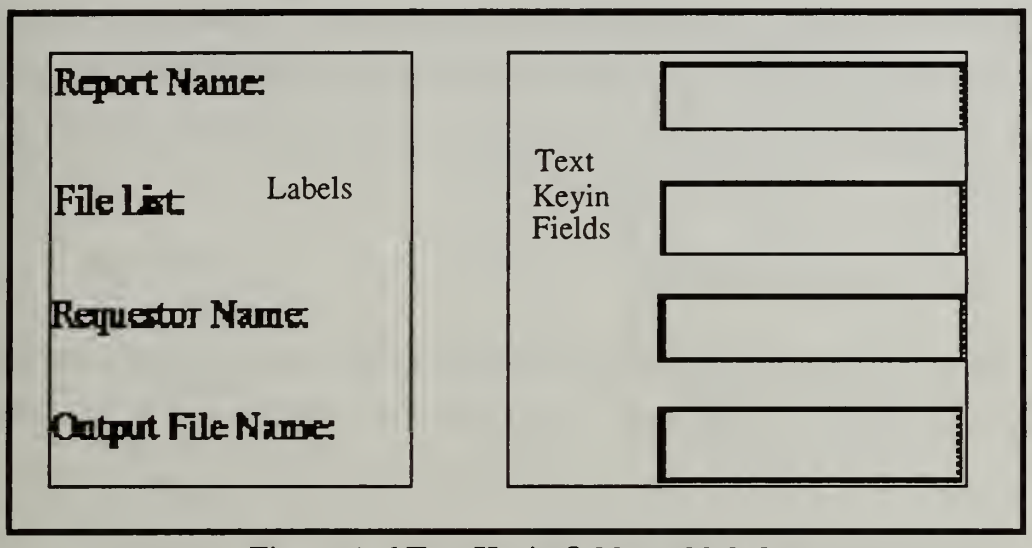

Figure A-6 Text Keyin fields and labels

E. INTRODUCTORY PANEL

The first panel is an introduction panel. The introduction panel contains the name of the tool, name of the author and three push-buttons displayed along the bottom. Beside the quit and help push-buttons, the other push-button is the "Next screen" button. The next screen button will make the introduction panel disappear, and bring up the second panel. The next screen push-button is the default push-button for the introduction panel.

F. INPUT PANEL

The second panel is where the user will enter the mandatory information for the tool to operate. The second panel contains four string keyin areas, a group of six checkboxes and four push-buttons. The string keyin areas are for the report name, file list, requestor name and output file name, respectively. The user enters the appropriate information by placing the cursor over the window and type in the appropriate information. Six checkboxes represent six different reports that can be generated by this tool. Any one or all of the

checkboxes may be selected. Report A is the default selection. In addition to the quit and help push-button, the second panel also has displayed along the bottom a "Generate Report" push-button and a "Specify Custom Report" push-button. When the generate report button is pressed, the second panel will disappear and the generate report panel will appear on the screen. When the specify custom report push-button is selected, the second panel disappears, and activates the third panel. The generate report push-button is the default push-button for the second panel.

G. ATTRIBUTE PANELS

The third through the seventh panels contain the attributes of the SEI checklist, one attribute per panel. (See Figure A-7) (See Figure A-8) (See Figure A-9) (See Figure A-10) (See Figure A-11) There are five push-buttons on each of these panels. The first two push-buttons are the quit and help buttons. Another push-button is the "Previous Screen" push-button. When the previous screen button is pressed, the current panel disappears, and activates the previous panel. The next push-button is labeled "Next Screen". When pressed, the current panel will disappear, activating the next panel in the sequence. The last push-button is the generate report button. When this button is pressed it will make the current panel disappear and activate the generate report panel. The generate report panel is the default button. .

Along the top right corner in panels three through seven are two radio-buttons displayed, one above the other. These radio-buttons allow the user to specify that in addition to the individual totals, this attribute will be included in a multi-dimension array at the end of the normal report format. When this choice is selected, all of the attributes selected as such (must have at least two) will be displayed as two or three dimensional arrays at the end of report F. When more than three attributes are selected, then all combinations of N choose three will be displayed at the end of report F, where N will be either four or five. The other major part of panels three through seven are the radio-buttons that correspond to the values for each attribute. All of the radio-buttons are the same, either

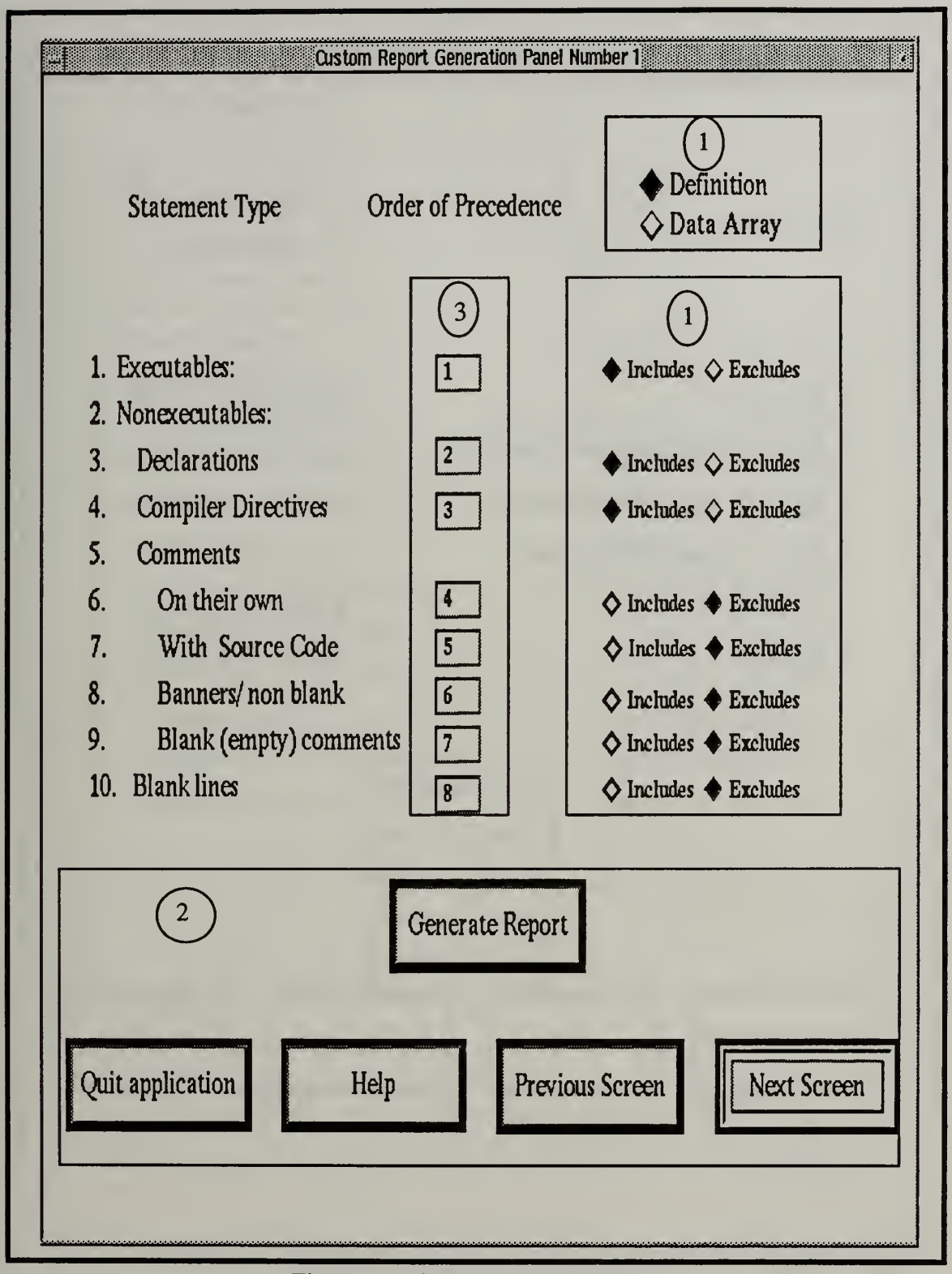

Figure A-7 Statement Type Panel

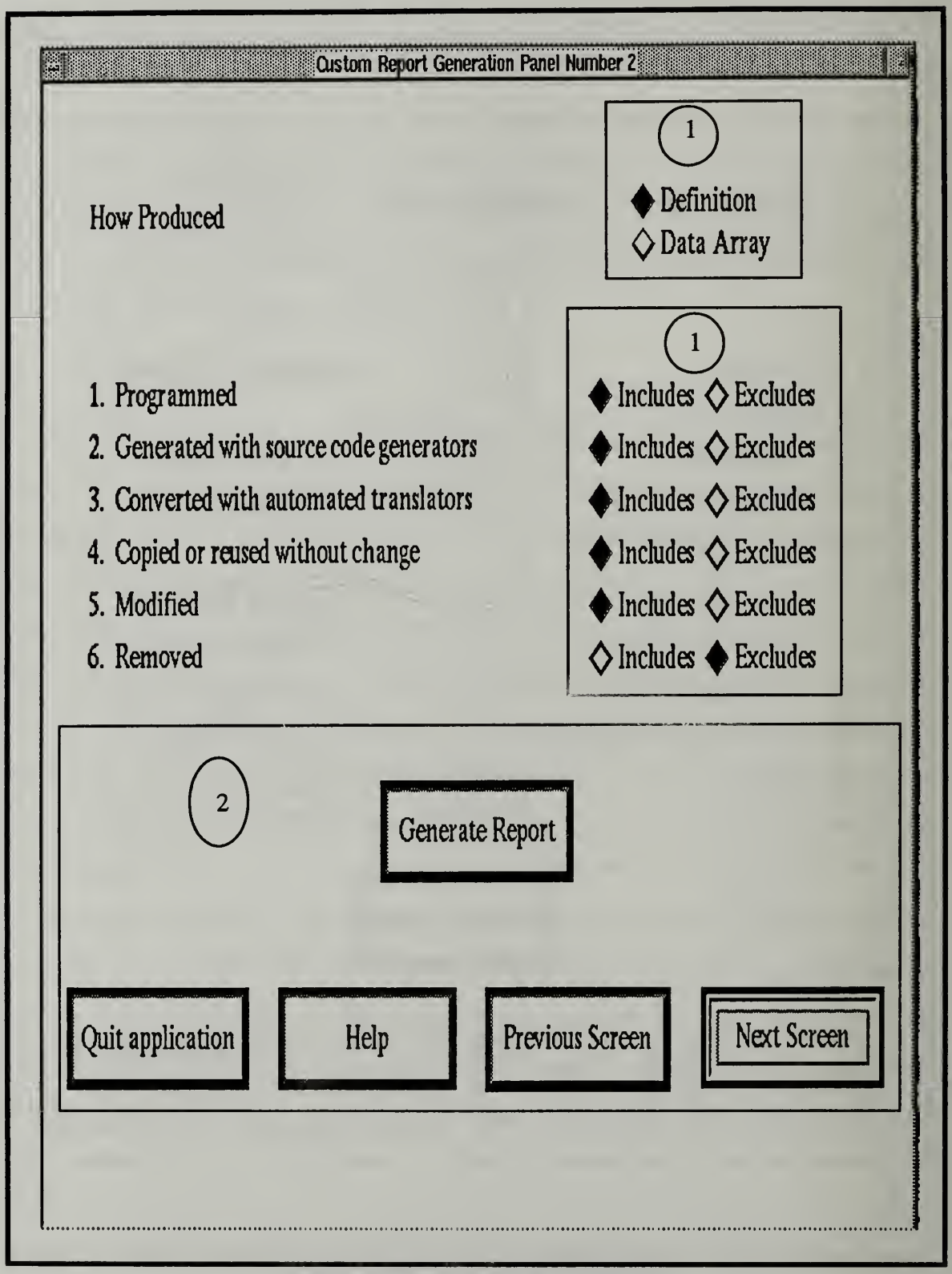

Figure A-8 How Produced Panel

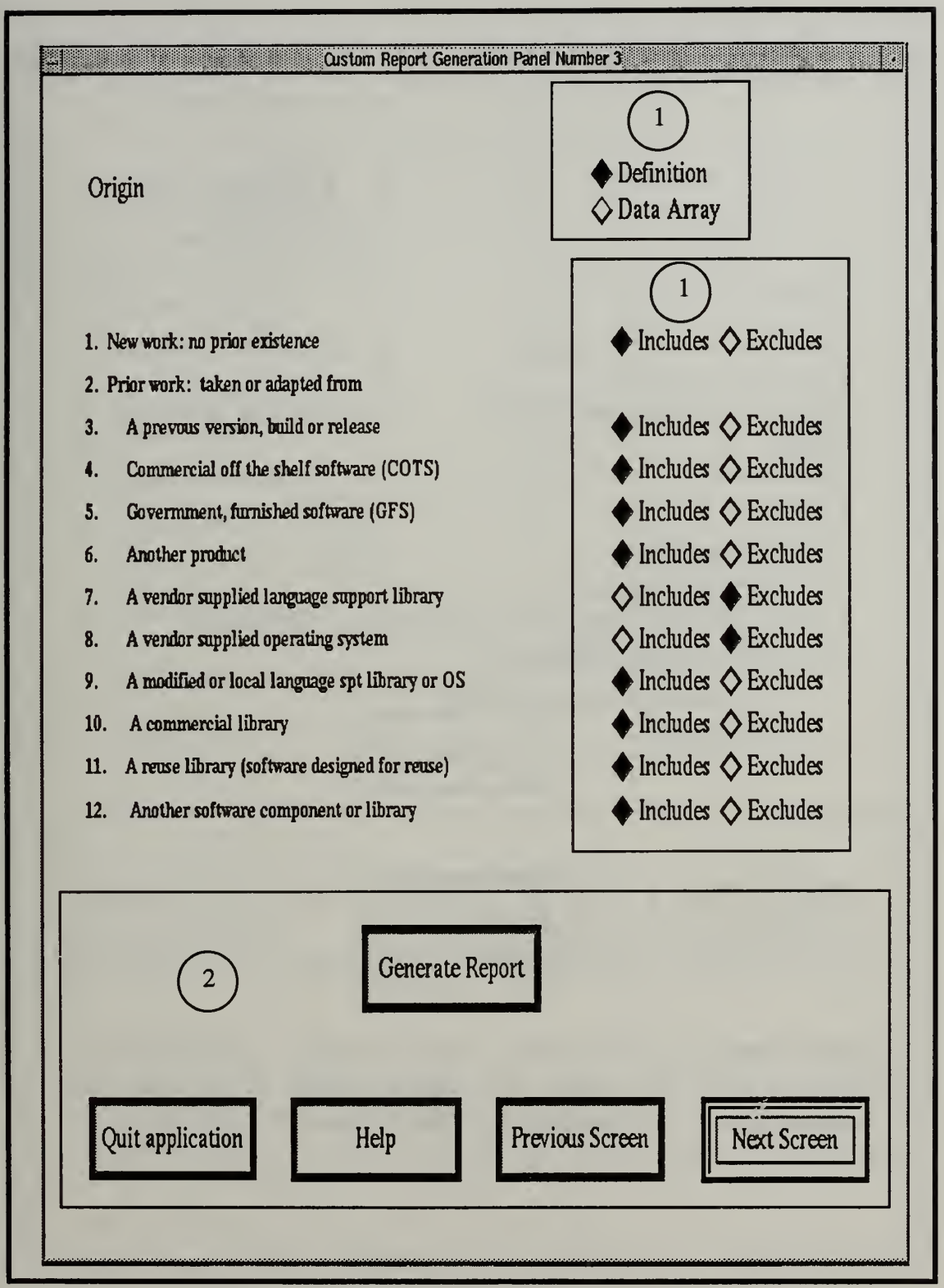

Figure A-9 Origin Panel

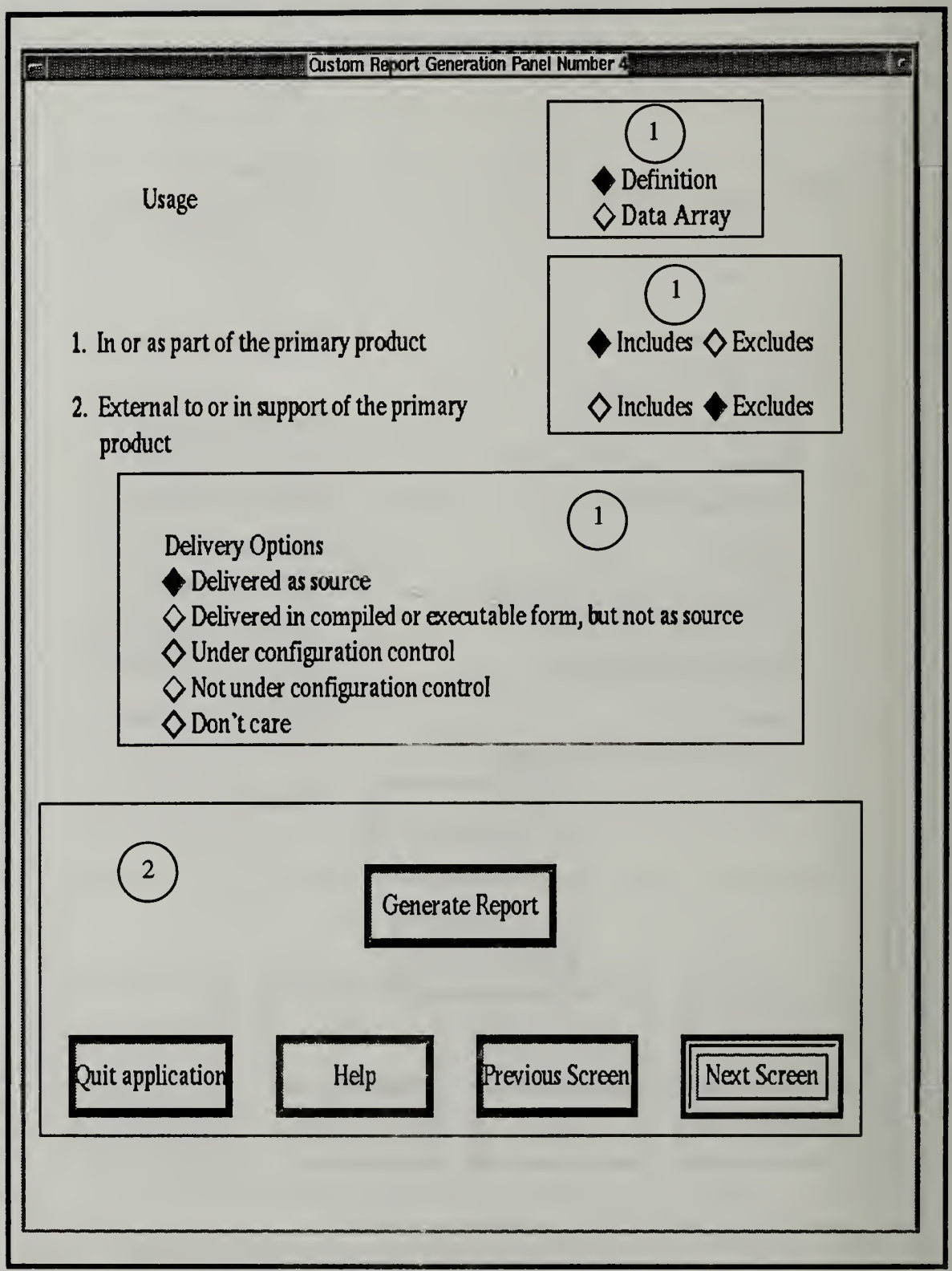

Figure A-10 Usage and Delivery Options Panel

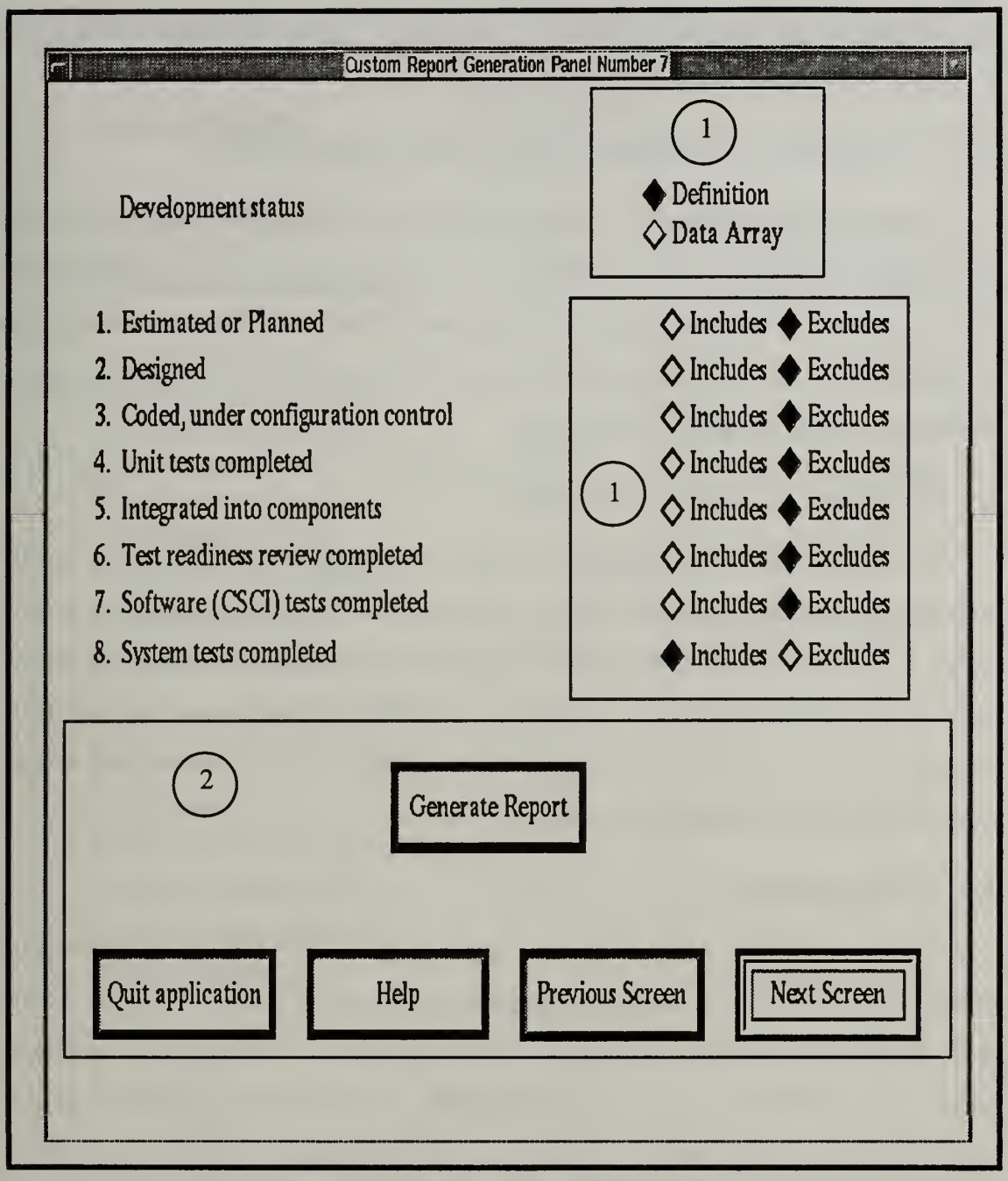

Figure A-11 Development Status Panel

the include button is highlighted or the exclude button is highlighted. Panel three also has eight integer keyin windows. Each integer keyin window corresponds to one of the values of the attribute statement type. TAE will ensure that the precedence entered is within the

range of one through eight. However, if the user does not ensure that each precedence value is unique, the results for report F may not be accurate.

H. CLARIFICATIONS (GENERAL and Ada) PANELS

The eight and ninth panels are for the general and Ada specific clarifications panels respectively. (See Figure A-12) (See Figure A-13) These two panels are similar to panel three, the difference being the number of radio-buttons. There are a total of thirteen general clarifications and six Ada specific clarifications. Each clarification is associated with one of the values of the attribute statement type.

I. GENERATE REPORT PANEL

The tenth panel is the generate report panel. (See Figure A-14) This panel has two push-buttons displayed along the bottom of the panel, they are labeled cancel and generate report. The generate report is the default button. The cancel button will make the generate report panel disappear. The user must then use the mouse to click on the icon of the previous panel. In addition to the push-buttons, this panel displays a text message explaining the different options available to the user.

J. QUIT PANEL

The last panel is the quit panel. This panel also has two push-buttons displayed along the bottom of the panel, they are labeled quit and cancel. (See Figure A-15) The quit button is the default button. When the quit button is selected, the panel will disappear and the tool will terminate. When the cancel button is selected, the quit panel disappears leaving the previous panel as the active panel. In addition to the push-buttons, there is a text area that displays the options to the user. This is provided in lieu of a help button.

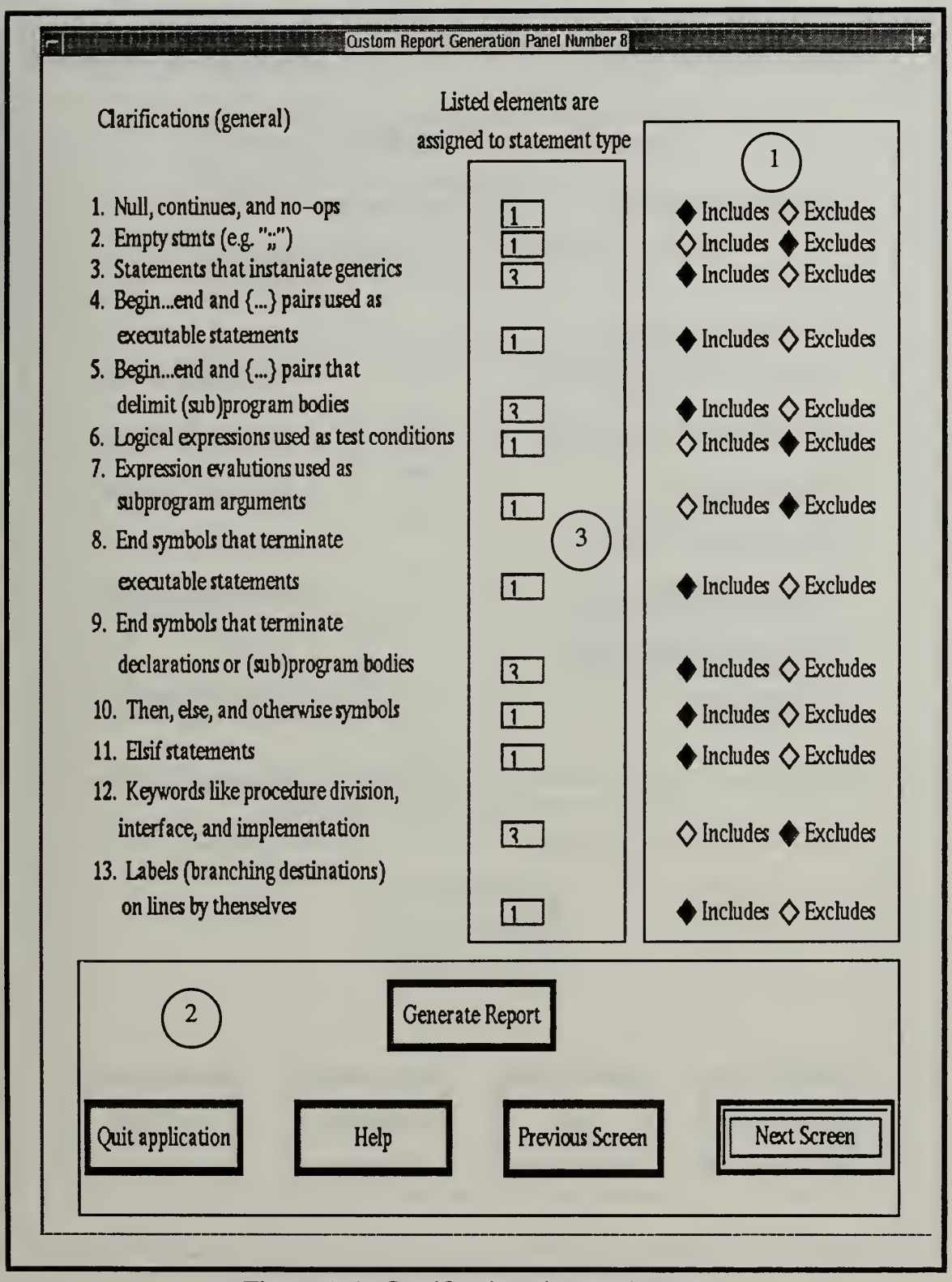

Figure A-12 Clarifications (general) Panel

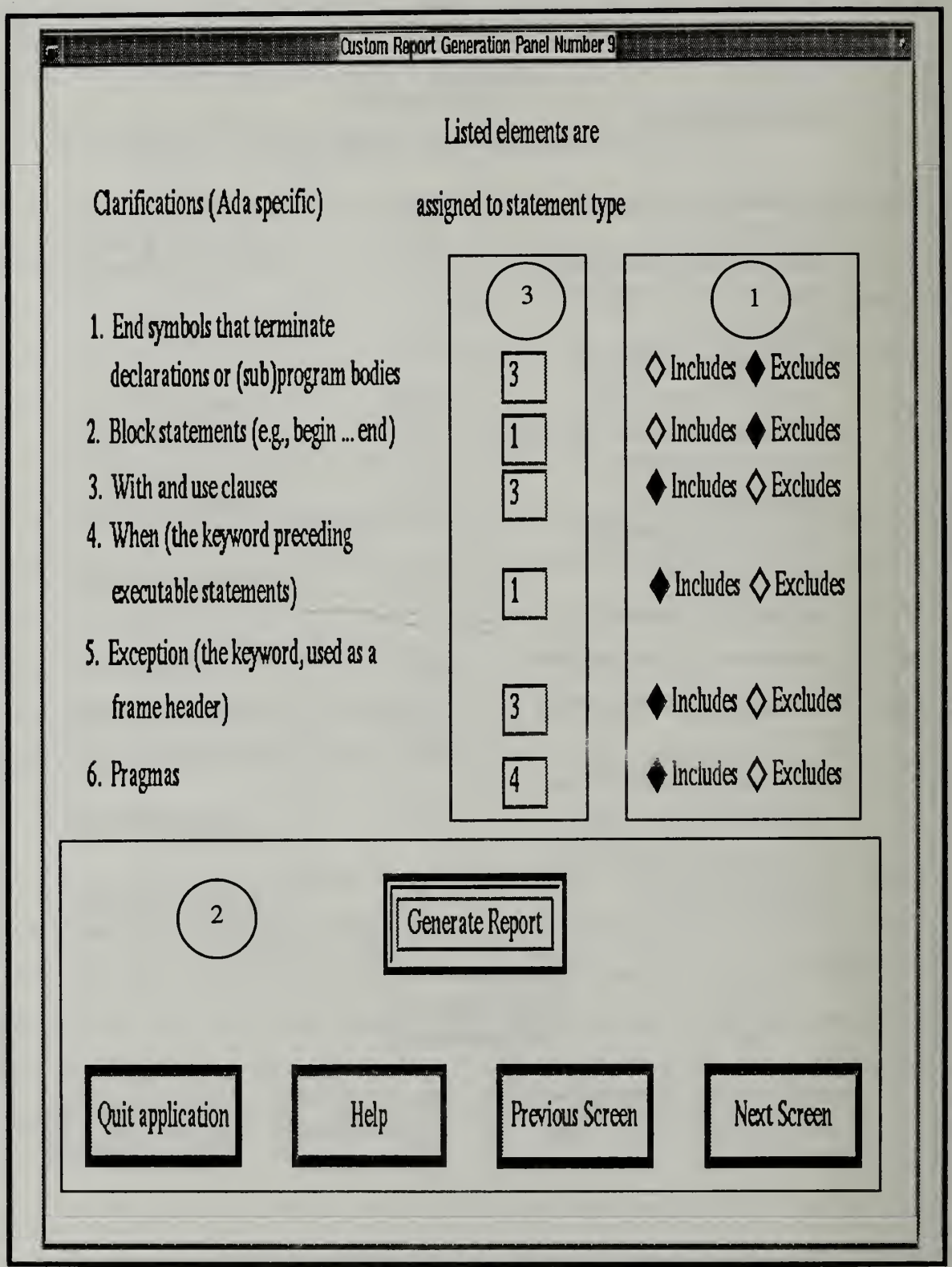

Figure A-13 Clarifications (Ada) Panel

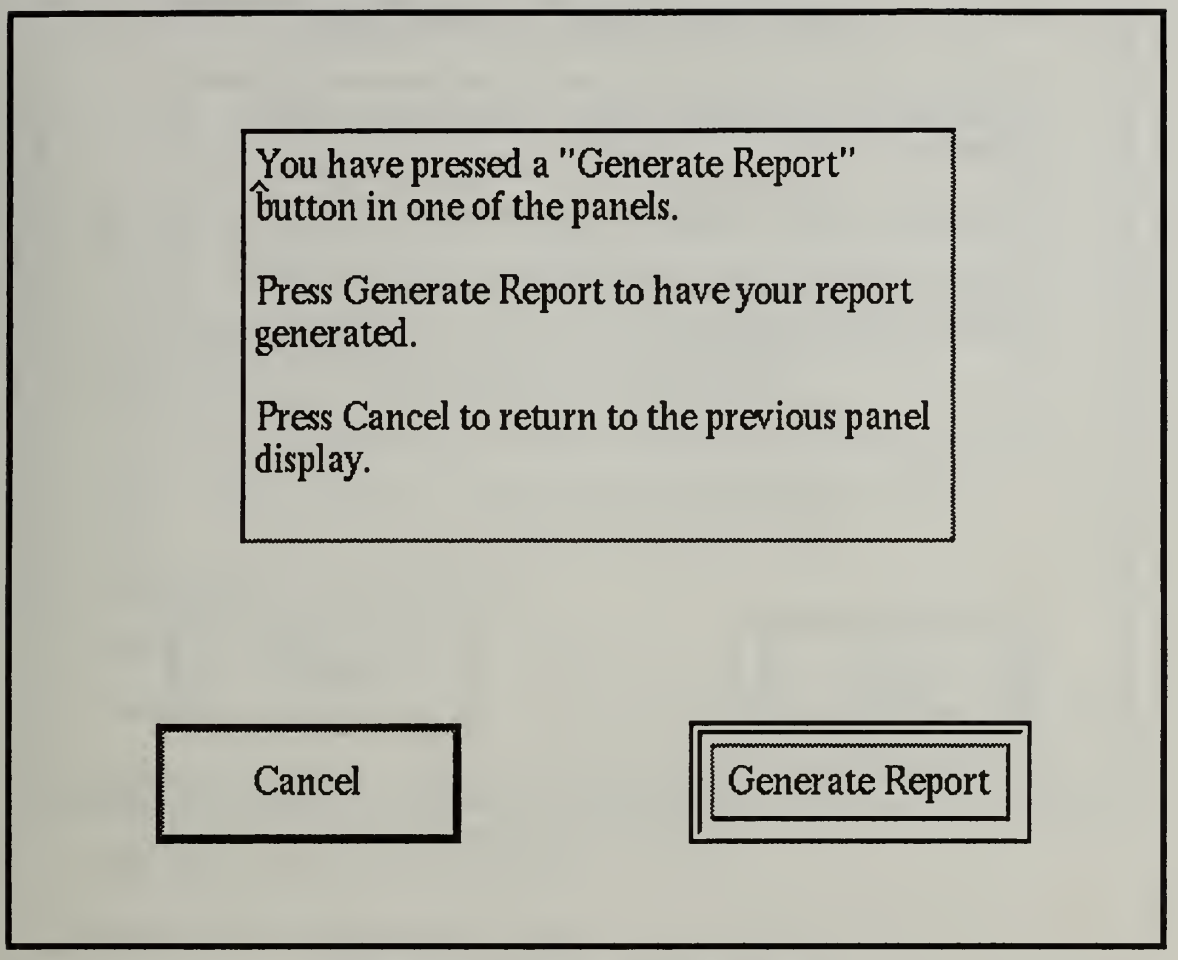

Figure A-14 Generate Report Panel

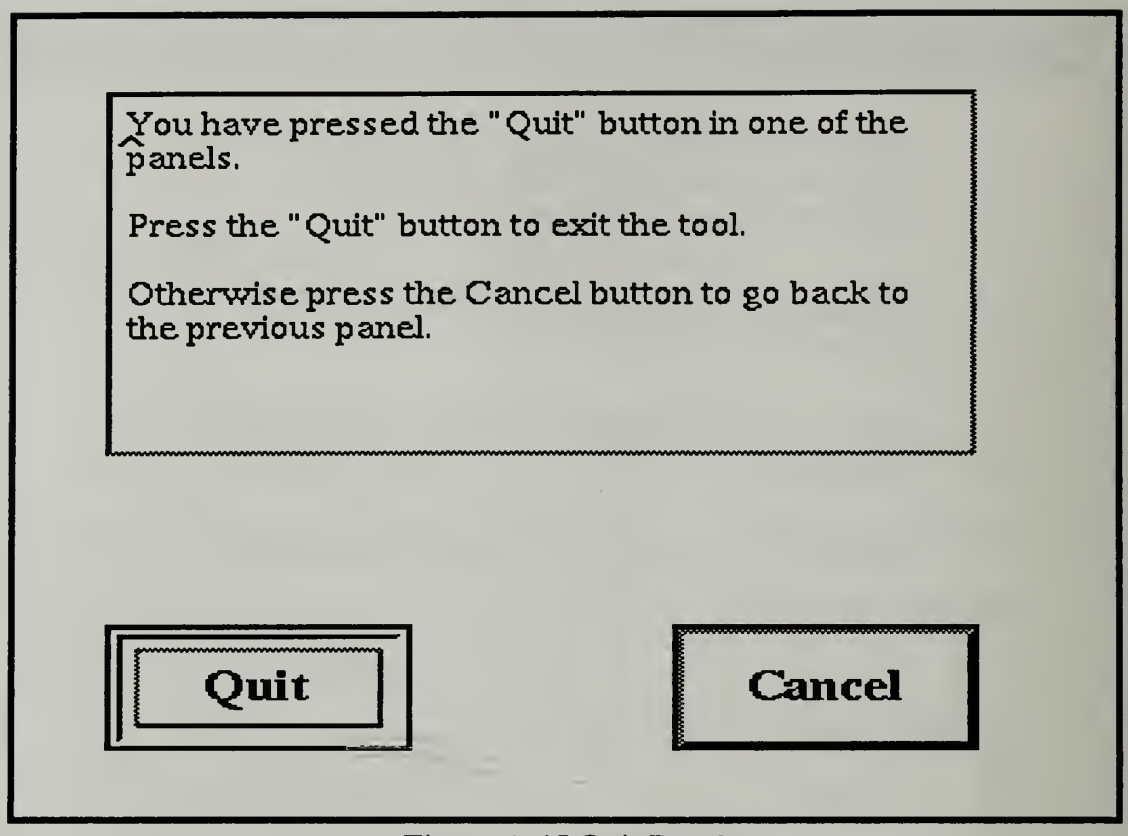

Figure A-15 Quit Panel

APPENDIX B. SOURCE CODE

ADA.Y

```
%token '&' ''' '(' ')' '*' '+' ',' '-' '.' '/' ':' ';'
%token '<' '=' '>' '|'

%token ARROW DOUBLE_DOT DOUBLE_STAR ASSIGNMENT INEQUALITY
%token GREATER_THAN_OR_EQUAL LESS_THAN_OR_EQUAL
%token LEFT_LABEL_BRACKET RIGHT_LABEL_BRACKET
%token BOX

%token ABORT_TOKEN ABS_TOKEN ACCEPT_TOKEN  ACCESS_TOKEN
%token ALL_TOKEN  AND_TOKEN ARRAY_TOKEN AT_TOKEN

%token BEGIN_TOKEN  BODY_TOKEN

%token CASE_TOKEN  CONSTANT_TOKEN

%token DECLARE_TOKEN DELAY_TOKEN DELTA_TOKEN DIGITS_TOKEN DO_TOKEN

%token ELSE_TOKEN ELSIF_TOKEN END_TOKEN  ENTRY_TOKEN EXCEPTION_TOKEN
%token EXIT_TOKEN

%token FOR_TOKEN FUNCTION_TOKEN

%token GENERIC_TOKEN  GOTO_TOKEN

%token IF_TOKEN IN_TOKEN  IS_TOKEN

%token LIMITED_TOKEN LOOP_TOKEN

%token MOD_TOKEN

%token NEW_TOKEN NOT_TOKEN  NULL_TOKEN

%token OF_TOKEN OR_TOKEN OTHERS_TOKEN OUT_TOKEN

%token PACKAGE_TOKEN PRAGMA_TOKEN  PRIVATE_TOKEN PROCEDURE_TOKEN

%token RAISE_TOKEN  RANGE_TOKEN RECORD_TOKEN REM_TOKEN
RENAMES_TOKEN
%token RETURN_TOKEN REVERSE_TOKEN

%token SELECT_TOKEN  SEPARATE_TOKEN SUBTYPE_TOKEN

%token TASK_TOKEN TERMINATE_TOKEN THEN_TOKEN TYPE_TOKEN
```

```
%token USE_TOKEN

%token WHEN_TOKEN WHILE_TOKEN WITH_TOKEN

%token XOR_TOKEN

%token IDENTIFIER
%token INTEGER_LITERAL REAL_LITERAL

%token CHARACTER_LITERAL STRING_LITERAL

%token ERROR1 ERROR2 ERROR3 ERROR4 ERROR5 ERROR6 ERROR7 ERROR8
%token ERROR9 ERROR10 ERROR11 ERROR12 ERROR13 ERROR14 ERROR15

%start compilation

{
  subtype yystype is integer;
}


%%

set_exec_start  :  {FLAGS_ARRAY(STMT_TYPE'val(0), 1) := true; };

set_exec_end    :  {FLAGS_ARRAY(STMT_TYPE'val(0), 2) := true; };

set_dec_start   :  {FLAGS_ARRAY(STMT_TYPE'val(1), 1) := true;
                   put (" dec start "); };

set_dec_end     :  {FLAGS_ARRAY(STMT_TYPE'val(1), 2) := true;
                   put (" dec end "); };

count_last_line :  { if DECLEVEL > 0 then
                       DECREASE_DECLEVEL;
                     end if;
                     new_line;
                     GLOBAL.ADD_TO_ARRAY;
                   } ;


task_body_or_body_stub  :
        check_task_token_body_token_sim_n
        check_task_body_or_body_stub
          ;


check_task_token_body_token_sim_n  :
          TASK_TOKEN
          BODY_TOKEN
      set_dec_start
```

```
        sim_n
        IS_TOKEN
      set_dec_end
    ;

check_task_body_or_body_stub    :
       SEPARATE_TOKEN
    |
      .decl_part.
    ;


check_package_body_or_body_stub :
     check_package_body_stub_common
     check_package_body_or_stub
     ;


check_package_body_stub_common :
        PACKAGE_TOKEN
        BODY_TOKEN
      set_dec_start
        sim_n
        IS_TOKEN
      set_dec_end  ;

check_package_body_or_stub    :
       SEPARATE_TOKEN
    |
      .decl_part.
    ;

-- Clarifications general

-- line 1
check_null_start : { if COUNT_CLARIFICATION
(GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_1) then
                 FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
                 put (" exec start");
                else
                 null;
                end if; } ;

check_null_end  : { if COUNT_CLARIFICATION
(GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_1) then
                 FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
                 put (" exec end");
                else
                 null;
                end if; } ;
```

```
-- line 2
-- Not applicable to Ada

-- line 3
-- expanded in-line
-- check_gen_inst_start
-- check_gen_inst_end

-- line 4
check_begin..end_start  :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_4) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
        put (" exec start ");
       else
        null;
       end if; } ;

check_begin..end_end   :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_4) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
        put (" exec end ");
       else
        null;
       end if; } ;

check_end_block_stmt  :
          END_TOKEN  check_begin..end_end ;

-- line 5
check_begin..end_delinate_start  :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_5) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
        put (" exec start ");
       else
        null;
       end if; } ;

check_begin..end_delinate_end   :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_5) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
        put (" exec end ");
       else
        null;
       end if; } ;

check_begin_stmt  :
      check_begin..end_delinate_start  BEGIN_TOKEN  check_begin..end_delinate_end ;

-- line 6
-- Not specific to Ada
```

```
-- line 7
-- Are considered part of Executable statement

-- line 8
check_end_exec_statement_start  :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_8) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
        put (" exec start ");
       else
        null;
       end if; } ;

check_end_exec_statement_end   :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_8) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
        put (" exec end ");
       else
        null;
       end if; } ;

CHECK_END_EXEC_STMT   :
          END_TOKEN  check_end_exec_statement_end  ;

-- line 9
check_end_declarations_start  :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_9) then
        FLAGS_ARRAY (STMT_TYPE'val (1), 1) := TRUE;
        put (" dec start ");
       else
        null;
       end if; } ;

check_end_declarations_end   :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_9) then
        FLAGS_ARRAY (STMT_TYPE'val (1), 2) := TRUE;
        put (" dec end ");
       else
        null;
       end if; } ;

check_end_dec    :
               END_TOKEN
              check_end_declarations_end
           ;

-- line 10
-- check for else, then, others on line by themselves
-- Is now tested for inside of ada_lex.l

-- line 11
-- check for elsif on line by itself
```

```
-- is now tested for inside of ada_lex.l

-- line 12
-- Does not apply to Ada

-- line 13
check_label_start   :
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_13) then
          FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
          put (" exec start ");
         else
          null;
         end if; }  ;

check_label_end    :
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_13) then
          FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
          put (" exec end ");
         else
          null;
         end if; }  ;


-- Ada specific clarifications

-- line 1
-- checkec in line 9 of general clarifications

-- line 2
-- checked in line 4 of general clarifications

-- line 3
check_with_and_use_start   :
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 1) := TRUE;
          put (" dec start ");
         else
          null;
         end if; }  ;

check_with_and_use_end    :
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 2) := TRUE;
          put (" dec end ");
         else
          null;
         end if; }  ;

-- line 4
check_when_start   :
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_4) then
```

```
        FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
        put (" exec start ");
       else
        null;
       end if; } ;

check_when_end    :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_4) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
        put (" exec end ");
       else
        null;
       end if; } ;


check_when       :
      check_when_start WHEN_TOKEN check_when_end ;

-- line 5
-- working
check_exception_keyword_start   :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_5) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
        put (" exec start ");
       else
        null;
       end if; } ;

check_exception_keyword_end    :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_5) then
        FLAGS_ARRAY (STMT_TYPE'val (0), 2) := TRUE;
        put (" exec end ");
       else
        null;
       end if; } ;

-- line 6
check_pragma_start   :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_6) then
        FLAGS_ARRAY (STMT_TYPE'val (2), 1) := TRUE;
        put (" pragma start ");
       else
        null;
       end if; } ;

check_pragma_end    :
      { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL11.LINE_6) then
        FLAGS_ARRAY (STMT_TYPE'val (2), 2) := TRUE;
        put (" pragma end ");
       else
        null;
```

```
        end if; } ;

--
-- Ayacc grammer rules follow
--

prag: check_pragma_start
  PRAGMA_TOKEN IDENTIFIER .arg_ascs ';'
        check_pragma_end ;


--prag: PRAGMA_TOKEN IDENTIFIER .arg_ascs ';' ;


arg_asc     :
  expr
  |IDENTIFIER ARROW expr ;


-- *** Added    *** --
numeric_literal
  : REAL_LITERAL
  | INTEGER_LITERAL
  ;



basic_d :
      object_d  set_dec_end
  |set_dec_start ty_d      set_dec_end
  |  subty_d   set_dec_end
  |subprg_d
  |  pkg_d
  |task_d    set_dec_end
  |  gen_d     set_dec_end
  |excptn_d  set_dec_end
  |  gen_inst
  |  renaming_d set_dec_end
  |  number_d  set_dec_end
  |  error ';' ;



object_d :
    set_dec_start idents ':'         subty_ind ._ASN_expr. ';'
|  set_dec_start idents ':'  CONSTANT_TOKEN subty_ind ._ASN_expr. ';'
|  set_dec_start idents ':'         c_arr_def ._ASN_expr. ';'
|  set_dec_start idents ':'  CONSTANT_TOKEN c_arr_def ._ASN_expr. ';' ;


number_d :
```

```
  set_dec_start idents ':' CONSTANT_TOKEN ASSIGNMENT expr ';' ;


idents : IDENTIFIER ...ident..;



ty_d :
  full_ty_d
  |incomplete_ty_d
  |priv_ty_d ;


full_ty_d :
    TYPE_TOKEN IDENTIFIER          IS_TOKEN ty_def ';'
  |
    TYPE_TOKEN IDENTIFIER discr_part IS_TOKEN ty_def ';' ;


ty_def :
  enum_ty_def| integer_ty_def
  |real_ty_def| array_ty_def
  |rec_ty_def | access_ty_def
  |derived_ty_def ;



subty_d :
  set_dec_start SUBTYPE_TOKEN IDENTIFIER IS_TOKEN subty_ind ';' ;


subty_ind :ty_mk .constrt. ;


ty_mk : expanded_n ;


constrt :
  rng_c
  |fltg_point_c | fixed_point_c
  |aggr ;



derived_ty_def : NEW_TOKEN subty_ind ;


rng_c : RANGE_TOKEN rng ;
```

```
rng :
  name
  lsim_expr DOUBLE_DOT sim_expr;


enum_ty_def    :
  '(' enum_lit_spec
        ...enum_lit_spec.. ')' ;

enum_lit_spec : enum_lit ;

enum_lit : IDENTIFIER | CHARACTER_LITERAL ;


integer_ty_def : rng_c ;


real_ty_def    :
  fltg_point_c | fixed_point_c ;


fltg_point_c :
  fltg_accuracy_def .rng_c. ;

fltg_accuracy_def :
  DIGITS_TOKEN sim_expr ;


fixed_point_c:
  fixed_accuracy_def .rng_c. ;

fixed_accuracy_def :
  DELTA_TOKEN sim_expr ;


array_ty_def    :
  uncnstrnd_array_defl c_arr_def ;

uncnstrnd_array_def:
```

```
    ARRAY_TOKEN '(' idx_subty_def ...idx_subty_def.. ')' OF_TOKEN
            subty_ind ;


c_arr_def    :
  ARRAY_TOKEN idx_c OF_TOKEN subty_ind ;


idx_subty_def    :    name  RANGE_TOKEN BOX ;


idx_c  :  '(' dscr_rng ...dscr_rng.. ')' ;


dscr_rng:
  rng
  lname rng_c;


rec_ty_def :
  RECORD_TOKEN
     cmpons
  CHECK_END_DEC  RECORD_TOKEN ;


--rec_ty_def :
--RECORD_TOKEN
--    cmpons
--END_TOKEN RECORD_TOKEN ;


cmpons:
  ..prag.. ..cmpon_d.. cmpon_d ..prag..
  |..prag.. ..cmpon_d.. variant_part ..prag..
  |   ..prag.. NULL_TOKEN ';' ..prag.. ;


cmpon_d  :  set_dec_start
    idents ':' cmpon_subty_def ._ASN_expr. ';' set_dec_end ;


cmpon_subty_def     :  subty_ind ;




discr_part  :
      '(' discr_spec ...discr_spec.. ')' ;


discr_spec  :
```

```
  idents ':' ty_mk ._ASN_expr. ;



variant_part :
  CASE_TOKEN sim_n IS_TOKEN
      ..prag.. variant ..variant..
    CHECK_END_DEC  CASE_TOKEN ';' ;


--variant_part :
--CASE_TOKEN sim_n IS_TOKEN
--     ..prag.. variant ..variant..
--      END_TOKEN CASE_TOKEN ';' ;


variant :
  CHECK_WHEN choice ..or_choice.. ARROW
     cmpons ;

--variant :
--WHEN_TOKEN choice ..or_choice.. ARROW
--    cmpons ;

choice : sim_expr
  | name rng_c
  | sim_expr DOUBLE_DOT sim_expr
  | OTHERS_TOKEN
  | error
  ;


access_ty_def :  ACCESS_TOKEN subty_ind ;


incomplete_ty_d :
      TYPE_TOKEN IDENTIFIER ';'
  |
      TYPE_TOKEN IDENTIFIER discr_part ';'  ;


decl_part :
  ..basic_decl_item..
  |   ..basic_decl_item.. body ..later_decl_item.. ;

basic_decl_item :
  basic_d
  |rep_cl  |  use_cl;
```

```
later_decl_item : body
  |subprg_d set_dec_end
  | pkg_d set_dec_end
  |task_dset_dec_end
  | gen_d set_dec_end
  |use_cl
  | gen_inst ;

body : proper_body | body_stub ;


proper_body :
  subprg_body | pkg_body | task_body ;


name : sim_n
  | CHARACTER_LITERAL | op_symbol
  |idxed_cmpon
  |selected_cmpon | attribute ;


sim_n :IDENTIFIER ;


prefix: name ;



idxed_cmpon :
  prefix aggr ;


selected_cmpon : prefix '.' selector ;


selector : sim_n
  | CHARACTER_LITERAL | op_symbol | ALL_TOKEN ;


attribute : prefix ''' attribute_designator ;


attribute_designator :
  sim_n
  |DIGITS_TOKEN
  |DELTA_TOKEN
  |RANGE_TOKEN ;


aggr :
     '(' cmpon_asc ...cmpon_asc.. ')' ;
```

```
cmpon_asc      :
  expr
  |choice ..or_choice.. ARROW expr
  |sim_expr DOUBLE_DOT sim_expr
  |name rng_c ;


expr :
  rel..AND__rel.. | rel..AND__THEN__rel..
  |rel..OR__rel..| rel..OR__ELSE__rel..
  |rel..XOR__rel.. ;


rel :
  sim_expr .relal_op__sim_expr.
  |sim_expr.NOT.IN__rng_or_sim_expr.NOT.IN__ty_mk ;


sim_expr :
  .unary_add_op.term..binary_add_op__term.. ;


term : factor..mult_op__factor.. ;


factor: pri ._EXP___pri. |ABS_TOKEN pri |NOT_TOKEN pri ;


pri :
  numeric_literal  | NULL_TOKEN
  |allocator | qualified_expr
  |name
  |aggr ;


relal_op : '='
   | INEQUALITY
   | '<'
   | LESS_THAN_OR_EQUAL
   | '>'
   | GREATER_THAN_OR_EQUAL ;


binary_add_op : '+' | '-' | '&' ;


unary_add_op : '+' | '-' ;
```

```
mult_op : '*' | '/' | MOD_TOKEN | REM_TOKEN ;


qualified_expr:
  ty_mkaggr_or_ty_mkPexprP_ ;



allocator :
  NEW_TOKEN ty_mk
  |NEW_TOKEN ty_mk aggr
  |  NEW_TOKEN ty_mk ''' aggr ;


seq_of_stmts: ..prag.. stmt ..stmt.. { null; } -- Because of bug
        ;


stmt :
  ..label.. sim_stmt
  | ..label.. compound_stmt
  |  error  ';' ;


--stmt :
--..label.. sim_stmt
-- | ..label.. compound_stmt
-- |  error  ';' ;


sim_stmt :null_stmt
  |set_exec_start assignment_stmt set_exec_end
  |  set_exec_start exit_stmt     set_exec_end
  |set_exec_start return_stmt    set_exec_end
  |  set_exec_start goto_stmt     set_exec_end
  |set_exec_start delay_stmt     set_exec_end
  |  set_exec_start abort_stmt    set_exec_end
  |set_exec_start raise_stmt     set_exec_end
  |  set_exec_start code_stmt     set_exec_end
  |  set_exec_start name ';'      set_exec_end  ;


--sim_stmt :null_stmt
--  |assignment_stmt |  exit_stmt
--  |return_stmt    | goto_stmt
--  |delay_stmt     | abort_stmt
--  |raise_stmt     | code_stmt
--  | name ';'  ;
```

```
compound_stmt :
  set_exec_start if_stmt
  | set_exec_start case_stmt
  |set_exec_start loop_stmt
  | set_exec_start block_stmt
  |set_exec_start accept_stmt set_exec_end
  |set_exec_start select_stmt set_exec_end  ;


--compound_stmt :
--if_stmt
--  | case_stmt
--  |loop_stmt
--  | block_stmt
--  |accept_stmt
--  |select_stmt ;


label :
      check_label_start
       LEFT_LABEL_BRACKET sim_n RIGHT_LABEL_BRACKET
      check_label_end
        ;


null_stmt : check_null_start NULL_TOKEN ';' check_null_end ;


-- null_stmt : NULL_TOKEN ';' ;


assignment_stmt : name ASSIGNMENT expr ';' ;



if_stmt :
  IF_TOKEN cond THEN_TOKEN
     seq_of_stmts
  ..ELSIF__cond__THEN__seq_of_stmts..
  .ELSE__seq_of_stmts.
    CHECK_END_EXEC_STMT  IF_TOKEN ';' ;


--if_stmt :
--IF_TOKEN cond THEN_TOKEN
--   seq_of_stmts
--..ELSIF__cond__THEN__seq_of_stmts..
--.ELSE__seq_of_stmts.
--      END_TOKEN IF_TOKEN ';' ;
```

```
cond : expr ;


case_stmt:
  CASE_TOKEN expr IS_TOKEN
     case_stmt_alt..case_stmt_alt..
      CHECK_END_EXEC_STMT  CASE_TOKEN ';' ;

--case_stmt:
--CASE_TOKEN expr IS_TOKEN
--    case_stmt_alt..case_stmt_alt..
--       END_TOKEN CASE_TOKEN ';' ;


case_stmt_alt :
  CHECK_WHEN choice ..or_choice.. ARROW
     seq_of_stmts ;


--case_stmt_alt :
--WHEN_TOKEN choice ..or_choice.. ARROW
--    seq_of_stmts ;


loop_stmt:
  .sim_nC.
     .iteration_scheme. LOOP_TOKEN
        seq_of_stmts
      CHECK_END_EXEC_STMT  LOOP_TOKEN .sim_n. ';'  ;

--loop_stmt:
--.sim_nC.
--    .iteration_scheme. LOOP_TOKEN
--       seq_of_stmts
--         END_TOKEN LOOP_TOKEN .sim_n. ';' ;



iteration_scheme
  : WHILE_TOKEN cond
  | WHILE_TOKEN error
  | FOR_TOKEN loop_prm_spec
  | FOR_TOKEN error
  ;


loop_prm_spec :
  IDENTIFIER IN_TOKEN .REVERSE. dscr_rng ;
```

```
block_stmt :
  .sim_nC.
     .DECLARE__decl_part.
      check_begin..end_start
     BEGIN_TOKEN
      check_begin..end_start
        seq_of_stmts
     .EXCEPTION__excptn_handler..excptn_handler...
      check_end_block_stmt .sim_n. ';' ;

--block_stmt :
--.sim_nC.
--   .DECLARE__decl_part.
--   BEGIN_TOKEN
--      seq_of_stmts
--   .EXCEPTION__excptn_handler..excptn_handler...
--       END_TOKEN .sim_n. ';' ;



exit_stmt:
     EXIT_TOKEN .expanded_n. .WHEN__cond. ';' ;



return_stmt : RETURN_TOKEN .expr. ';' ;



goto_stmt : GOTO_TOKEN expanded_n ';' ;



subprg_d : subprg_spec ';' ;

procedure_ident :
  set_dec_start
  PROCEDURE_TOKEN IDENTIFIER  ;

function_desig :
     set_dec_start
  FUNCTION_TOKEN designator  ;

--function_desig :
--set_dec_start
--FUNCTION_TOKEN designator  set_dec_end ;
```

```
subprg_spec :
  procedure_ident .fml_part. set_dec_end
  |  function_desig  .fml_part. RETURN_TOKEN ty_mk  set_dec_end ;


designator : IDENTIFIER | op_symbol ;


op_symbol : STRING_LITERAL ;


fml_part :
      '(' prm_spec .._.prm_spec.. ')' ;


prm_spec :
  idents ':' mode ty_mk ._ASN_expr. ;


mode : .IN. | IN_TOKEN OUT_TOKEN | OUT_TOKEN  ;




subprg_body :
  subprg_spec IS_TOKEN
     .decl_part.
  check_begin..end_delinate_start
  BEGIN_TOKEN
  check_begin..end_delinate_end
     seq_of_stmts
  .EXCEPTION__excptn_handler..excptn_handler...
  check_begin..end_delinate_start
      END_TOKEN .designator. ';'
  check_begin..end_delinate_end
    ;

--subprg_body :
--subprg_spec IS_TOKEN
--    .decl_part.
--BEGIN_TOKEN
--    seq_of_stmts
--.EXCEPTION__excptn_handler..excptn_handler...
--      END_TOKEN  .designator. ';' ;


pkg_d : pkg_spec ';' ;

package_ident     :
                PACKAGE_TOKEN
                IDENTIFIER
```

```
                IS_TOKEN
            ;

set_dec_start_end :
                { FLAGS_ARRAY(STMT_TYPE'val(1), 1) := true;
                 put (" dec start pkg ");
                 FLAGS_ARRAY(STMT_TYPE'val(1), 2) := true;
                 put (" dec end pkg ");
                 }
            ;


--check_pkg_declaration  :
--                  package_ident
--                  set_dec_start_end
--                  IS_TOKEN
--               ;

check_pkg_declaration  :
                  PACKAGE_TOKEN
                  IDENTIFIER
                  set_dec_start
                  IS_TOKEN
                  set_dec_end
               ;

pkg_spec :
        check_pkg_declaration
      ..basic_decl_item..
      .PRIVATE..basic_decl_item...
        check_end_declarations_start
        check_end_dec
          .sim_n.
      ;

--pkg_spec :
--       PACKAGE_TOKEN
--IDENTIFIER IS_TOKEN
--   ..basic_decl_item..
--.PRIVATE..basic_decl_item...
--END_TOKEN .sim_n.    ;


pkg_body :
        check_package_body_or_body_stub
    .BEGIN__seq_of_stmts.EXCEPTION__excptn_handler..excptn_handler...
   check_begin..end_delinate_start
  END_TOKEN .sim_n. ';'
   check_begin..end_delinate_end
      ;
```

```
--pkg_body :
--PACKAGE_TOKEN BODY_TOKEN sim_n IS_TOKEN
--   .decl_part.
--   .BEGIN__seq_of_stmts.EXCEPTION__excptn_handler..excptn_handler...
--END_TOKEN .sim_n. ';'  ;


priv_ty_d :
    TYPE_TOKEN IDENTIFIER          IS_TOKEN .LIMITED. PRIVATE_TOKEN ';'
  |
    TYPE_TOKEN IDENTIFIER discr_part IS_TOKEN .LIMITED. PRIVATE_TOKEN ';' ;



use_cl : check_with_and_use_start
         USE_TOKEN expanded_n ...expanded_n.. ';'
       check_with_and_use_end ;


--renaming_d :
--       set_dec_start idents ':' ty_mk     RENAMES_TOKEN name ';'
-- | set_dec_start idents ':' EXCEPTION_TOKEN     RENAMES_TOKEN expanded_n ';'
-- |   package_ident  RENAMES_TOKEN expanded_n ';'
-- |   subprg_spec  RENAMES_TOKEN name ';'  ;


renaming_d :
      set_dec_start idents ':' ty_mk     RENAMES_TOKEN name ';'
 | set_dec_start idents ':' EXCEPTION_TOKEN     RENAMES_TOKEN expanded_n ';'
 |   PACKAGE_TOKEN IDENTIFIER       RENAMES_TOKEN expanded_n ';'
 |   subprg_spec  RENAMES_TOKEN name ';'  ;


task_d : task_spec ';' ;


task_spec :
  TASK_TOKEN set_dec_start .TYPE. IDENTIFIER
     .IS..ent_d_..rep_cl_END.sim_n.   ;


task_body :
    task_body_or_body_stub
    CHECK_BEGIN_STMT     seq_of_stmts
  .EXCEPTION__excptn_handler..excptn_handler...
    check_begin..end_delinate_start
  END_TOKEN .sim_n. ';'
    check_begin..end_delinate_end       ;
```

```
--task_body :
--TASK_TOKEN BODY_TOKEN sim_n IS_TOKEN
--   .decl_part.
--BEGIN_TOKEN
--   seq_of_stmts
--.EXCEPTION__excptn_handler..excptn_handler...
--      END_TOKEN .sim_n. ';' ;


ent_d :
  set_dec_start ENTRY_TOKEN IDENTIFIER .fml_part. ';' set_dec_end
  |  set_dec_start ENTRY_TOKEN IDENTIFIER '(' dscr_rng ')' .fml_part. ';'
     set_dec_end ;


ent_call_stmt :
     ..prag.. name ';' ;


accept_stmt :
  ACCEPT_TOKEN sim_n .Pent_idx_P..fml_part.
       .DO__seq_of_stmts__END.sim_n.. ';' ;


ent_idx :expr  ;



delay_stmt : DELAY_TOKEN sim_expr ';' ;



select_stmt :selec_wait
  lcondal_ent_calll timed_ent_call ;



selec_wait:
  SELECT_TOKEN
     select_alt
     ..OR__select_alt..
     .ELSE__seq_of_stmts.
      END_TOKEN  SELECT_TOKEN ';' ;


--selec_wait:
--SELECT_TOKEN
--   select_alt
--   ..OR__select_alt..
--   .ELSE__seq_of_stmts.
--      END_TOKEN SELECT_TOKEN ';' ;
```

```
select_alt :
  .WHEN__condARROW.selec_wait_alt ;


selec_wait_alt : accept_alt
  |delay_alt | terminate_alt ;


accept_alt :
  accept_stmt.seq_of_stmts. ;


delay_alt :
  delay_stmt.seq_of_stmts. ;


terminate_alt : TERM_stmt  ;


condal_ent_call:
  SELECT_TOKEN
     ent_call_stmt
     .seq_of_stmts.
  ELSE_TOKEN
     seq_of_stmts
      END_TOKEN   SELECT_TOKEN ';' ;


--condal_ent_call:
--SELECT_TOKEN
--    ent_call_stmt
--    .seq_of_stmts.
--ELSE_TOKEN
--    seq_of_stmts
--       END_TOKEN SELECT_TOKEN ';' ;



timed_ent_call :
  SELECT_TOKEN
     ent_call_stmt
     .seq_of_stmts.
  OR_TOKEN
     delay_alt
      END_TOKEN  SELECT_TOKEN ';' ;


--timed_ent_call :
--SELECT_TOKEN
```

```
--   ent_call_stmt
--   .seq_of_stmts.
--OR_TOKEN
--   delay_alt
--      END_TOKEN SELECT_TOKEN ';' ;


abort_stmt : ABORT_TOKEN name ...name.. ';' ;


compilation :..compilation_unit.. count_last_line ;


--compilation :..compilation_unit.. ;


compilation_unit :
  context_cl library_unit
  | context_cl secondary_unit ;

library_unit :
  subprg_dl pkg_d
    | gen_dl gen_inst
  |subprg_body   ;


secondary_unit:
  library_unit_body | subunit;


library_unit_body :
  pkg_body_or_subprg_body ;



context_cl    : ..with_cl..use_cl.... ;

with_cl : check_with_and_use_start
          WITH_TOKEN sim_n ...sim_n.. ';'
        check_with_and_use_end ;

--with_cl : set_dec_start WITH_TOKEN sim_n ...sim_n.. ';' set_dec_end ;


body_stub :
     subprg_spec IS_TOKEN SEPARATE_TOKEN ';'
  | check_package_body_or_body_stub  set_dec_end ';'
  | task_body_or_body_stub          set_dec_end ';'
```

```
  ;


--body_stub :
--      subprg_spec IS_TOKEN SEPARATE_TOKEN ';'
-- |  PACKAGE_TOKEN BODY_TOKEN sim_n IS_TOKEN SEPARATE_TOKEN ';'
-- |  TASK_TOKEN BODY_TOKEN sim_n   IS_TOKEN SEPARATE_TOKEN ';'  ;


subunit : SEPARATE_TOKEN '(' expanded_n ')' proper_body ;




excptn_d : set_dec_start idents ':' EXCEPTION_TOKEN ';' ;


excptn_handler:
    CHECK_WHEN  excptn_choice ..or_excptn_choice.. ARROW
     seq_of_stmts ;

--excptn_handler:
--WHEN_TOKEN excptn_choice ..or_excptn_choice.. ARROW
--    seq_of_stmts ;


excptn_choice : expanded_n |OTHERS_TOKEN;


raise_stmt : RAISE_TOKEN .expanded_n. ';' ;


gen_d : gen_spec ';' ;


gen_spec :
  gen_fml_part   subprg_spec
  |gen_fml_part   pkg_spec ;


--gen_spec :
--gen_fml_part  subprg_spec
-- |gen_fml_part  pkg_spec ;


gen_fml_part :set_dec_start GENERIC_TOKEN set_dec_end ..gen_prm_d..  ;


--gen_fml_part :    GENERIC_TOKEN ..gen_prm_d..  ;
```

```
gen_prm_d :
  set_dec_start
        idents ':' .IN.OUT.. ty_mk ._ASN_expr. ';'
      set_dec_end
  |   set_dec_start
       TYPE_TOKEN IDENTIFIER IS_TOKEN gen_ty_def ';'
      set_dec_end
  |set_dec_start
       priv_ty_d
      set_dec_end
  |    WITH_TOKEN subprg_spec .IS_BOX_. ';'
       ;


--gen_prm_d :
--idents ':' .IN.OUT.. ty_mk ._ASN_expr. ';'
--  |   TYPE_TOKEN IDENTIFIER IS_TOKEN gen_ty_def ';'
--  |priv_ty_d
--  |   WITH_TOKEN subprg_spec .IS_BOX_. ';' ;


gen_ty_def   :
            '(' BOX ')'
          | RANGE_TOKEN BOX
          | DIGITS_TOKEN BOX
          | DELTA_TOKEN BOX
          | array_ty_def
          | access_ty_def
         ;


check_pkg_inst_declaration :
                    PACKAGE_TOKEN
                    IDENTIFIER
                    set_dec_start
                    IS_TOKEN
                    NEW_TOKEN
                  ;


gen_inst :
        check_pkg_inst_declaration
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 1) := TRUE;
          put (" dec start ");
         else
          null;
         end if; }
        expanded_n
        .gen_act_part.
      ';'
```

```
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 2) := TRUE;
          put (" dec end ");
         else
          null;
         end if; }
  |PROCEDURE__ident__IS_
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 1) := TRUE;
          put (" dec start ");
         else
          null;
         end if; }
       NEW_TOKEN expanded_n .gen_act_part. ';'
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 2) := TRUE;
          put (" dec end ");
         else
          null;
         end if; }
  |function_desig IS_TOKEN
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 1) := TRUE;
          put (" dec start ");
         else
          null;
        end if; }
    NEW_TOKEN expanded_n .gen_act_part. ';'
        { if COUNT_CLARIFICATION (GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_3) then
          FLAGS_ARRAY (STMT_TYPE'val (1), 2) := TRUE;
          put (" dec end ");
         else
          null;
         end if; }
     ;


gen_act_part :
          set_dec_start
           '(' gen_asc ...gen_asc.. ')'
          set_dec_end
        ;


--gen_act_part :
--      '(' gen_asc ...gen_asc.. ')'          ;


gen_asc    :
  .gen_fml_prmARROW.gen_act_prm ;
```

```
gen_fml_prm :
  sim_n | op_symbol ;

gen_act_prm :
 expr_or_name_or_subprg_n_or_ent_n_or_ty_mk
  ;


rep_cl :
  ty_rep_cl | address_cl ;

ty_rep_cl : length_cl
  lenum_rep_cl
  lrec_rep_cl   ;


length_cl : FOR_TOKEN attribute USE_TOKEN sim_expr ';' ;


enum_rep_cl :
     FOR__ty_sim_n__USE_ aggr ';' ;


rec_rep_cl:
  FOR__ty_sim_n__USE_
    RECORD_TOKEN .algt_cl.
       ..cmpon_cl..
     CHECK_END_DEC  RECORD_TOKEN ';'  ;

--rec_rep_cl:
--FOR__ty_sim_n__USE_
--   RECORD_TOKEN .algt_cl.
--       ..cmpon_cl..
--         END_TOKEN  RECORD_TOKEN ';'  ;


algt_cl :    AT_TOKEN MOD_TOKEN sim_expr ';' ;


cmpon_cl :
     name AT_TOKEN sim_expr RANGE_TOKEN rng ';' ;


address_cl : FOR_TOKEN sim_n USE_TOKEN AT_TOKEN sim_expr ';'  ;



code_stmt : ty_mk_rec_aggr ';' ;
```

```
..prag.. :
  |..prag.. prag ;

.arg_ascs :
  |  '(' arg_ascs ')' ;

arg_ascs :
  arg_asc
  |  arg_ascs ',' arg_asc ;


._ASN_expr.:
  |ASSIGNMENT expr ;

...ident.. :
  |  ...ident.. ',' IDENTIFIER  ;

.constrt. :
  |constrt  ;


expanded_n :
  IDENTIFIER
  |  expanded_n '.' IDENTIFIER  ;


...enum_lit_spec.. :
  |  ...enum_lit_spec.. ','
       enum_lit_spec ;

.rng_c. :
  |rng_c;

...idx_subty_def..      :
  |  ...idx_subty_def.. ',' idx_subty_def            ;


...dscr_rng.. :
  |  ...dscr_rng.. ',' dscr_rng ;


..cmpon_d.. :
  |  ..cmpon_d.. cmpon_d ..prag..   ;

...discr_spec..:
  |  ...discr_spec.. ';' discr_spec ;



..variant.. :
```

```
  |..variant.. variant ;

..or_choice.. :
  |  ..or_choice.. '|' choice ;



..basic_decl_item..  :
  ..prag..
  |..basic_decl_item.. basic_decl_item ..prag.. ;

..later_decl_item..  :
  ..prag..
  |  ..later_decl_item.. later_decl_item ..prag.. ;


...cmpon_asc..  :
  |  ...cmpon_asc.. ',' cmpon_asc ;



rel..AND__rel.. :
  rel AND_TOKEN rel
  |rel..AND__rel.. AND_TOKEN rel ;


rel..OR__rel.. :
  rel OR_TOKEN rel
  |rel..OR__rel.. OR_TOKEN rel ;


rel..XOR__rel.. :
  rel
  |..XOR__rel.. ;


..XOR__rel.. :
  rel XOR_TOKEN rel
  |..XOR__rel.. XOR_TOKEN rel;


rel..AND__THEN__rel.. :
  rel AND_TOKEN THEN_TOKEN rel
  |rel..AND__THEN__rel.. AND_TOKEN THEN_TOKEN rel ;


rel..OR__ELSE__rel..:
  rel OR_TOKEN ELSE_TOKEN rel
  |rel..OR__ELSE__rel.. OR_TOKEN ELSE_TOKEN rel ;
```

```
.relal_op__sim_expr. :
  |relal_op sim_expr ;


sim_expr.NOT.IN__rng_or_sim_expr.NOT.IN__ty_mk:
  sim_expr .NOT. IN_TOKEN rng ;


.NOT. :
  |NOT_TOKEN   ;

.unary_add_op.term..binary_add_op__term.. :
  term
  |unary_add_op term
  |.unary_add_op.term..binary_add_op__term..
  binary_add_op term  ;

factor..mult_op__factor..:
  factor
  |factor..mult_op__factor.. mult_op factor ;

._EXP___pri. :
  |DOUBLE_STAR pri ;



ty_mkaggr_or_ty_mkPexprP_ :
      prefix '' aggr ;


..stmt.. :
  ..prag..
  |..stmt.. stmt ..prag.. ;


..label..  :
  |..label.. label  ;



..ELSIF__cond__THEN__seq_of_stmts.. :
  |..ELSIF__cond__THEN__seq_of_stmts..
        ELSIF_TOKEN cond THEN_TOKEN
          seq_of_stmts   ;

.ELSE__seq_of_stmts.:
  |ELSE_TOKEN
     seq_of_stmts   ;

case_stmt_alt..case_stmt_alt.. :
     ..prag..
```

```
      case_stmt_alt
      ..case_stmt_alt.. ;

..case_stmt_alt..:
  |..case_stmt_alt.. case_stmt_alt ;

.sim_nC.:
  |   sim_n ':' ;

.sim_n. :
  |sim_n  ;

.iteration_scheme. :
  |iteration_scheme  ;

.REVERSE. :
  |REVERSE_TOKEN  ;

.DECLARE__decl_part.    :
  |set_exec_start DECLARE_TOKEN
     decl_partset_exec_end;


.EXCEPTION__excptn_handler..excptn_handler... :
  |check_exception_keyword_start
       EXCEPTION_TOKEN
     check_exception_keyword_end
     ..prag.. excptn_handlers  set_exec_end  ;


--.EXCEPTION__excptn_handler..excptn_handler... :
--  |EXCEPTION_TOKEN  ..prag.. excptn_handlers  set_exec_end  ;


excptn_handlers:
  excptn_handler
  |excptn_handlers excptn_handler   ;


.expanded_n. :
  |expanded_n  ;


.WHEN__cond. :
  |CHECK_WHEN   cond;


--.WHEN__cond. :
--  |WHEN_TOKEN cond;
```

```
.expr.:
  |expr  ;



.fml_part. :
  |fml_part ;

..._.prm_spec.. :
  |  ..._.prm_spec.. ';' prm_spec ;

.IN. :
  |IN_TOKEN ;

.decl_part.   : decl_part ;


.designator. :
  |  designator   ;



.PRIVATE..basic_decl_item... :
  |set_dec_start
       PRIVATE_TOKEN
     set_dec_end
    ..basic_decl_item.. ;


--.PRIVATE..basic_decl_item... :
--  |  PRIVATE_TOKEN
--    ..basic_decl_item.. ;


.BEGIN__seq_of_stmts.EXCEPTION__excptn_handler..excptn_handler...
            :
  !check_begin_stmt
     seq_of_stmts
  .EXCEPTION__excptn_handler..excptn_handler... ;


--.BEGIN__seq_of_stmts.EXCEPTION__excptn_handler..excptn_handler...
--            :
--  | BEGIN_TOKEN
--          seq_of_stmts
--.EXCEPTION__excptn_handler..excptn_handler... ;


.LIMITED. :
  |LIMITED_TOKEN   ;
```

```
...expanded_n.. :
  |   ...expanded_n.. ',' expanded_n  ;



.TYPE.:
  |TYPE_TOKEN  ;

.IS..ent_d_..rep_cl_END.sim_n. :
  |IS_TOKEN
     ..ent_d..
     ..rep_cl..
  END_TOKEN .sim_n.     ;


..ent_d.. :
  ..prag..
  |..ent_d.. ent_d ..prag..;

..rep_cl.. :
  |..rep_cl.. rep_cl ..prag..;


.Pent_idx_P..fml_part. :
  .fml_part.
  |   '(' ent_idx ')' .fml_part. ;

.DO__seq_of_stmts__END.sim_n.. :
  |DO_TOKEN
    seq_of_stmts
       END_TOKEN .sim_n. ;


--.DO__seq_of_stmts__END.sim_n.. :
--  |DO_TOKEN
-- seq_of_stmts
--END_TOKEN  .sim_n. ;


..OR__select_alt.. :
  |..OR__select_alt.. OR_TOKEN select_alt;


.WHEN__condARROW.selec_wait_alt :
  selec_wait_alt
  |CHECK_WHEN  cond ARROW selec_wait_alt  ;


--.WHEN__condARROW.selec_wait_alt :
--selec_wait_alt
--  |WHEN_TOKEN cond ARROW selec_wait_alt  ;
```

```
accept_stmt.seq_of_stmts. :
  ..prag.. accept_stmt .seq_of_stmts. ;

delay_stmt.seq_of_stmts. :
  ..prag.. delay_stmt .seq_of_stmts. ;


TERM_stmt  : ..prag..   TERMINATE_TOKEN ';'
           ..prag.. ;


--TERM_stmt  : ..prag.. set_exec_start  TERMINATE_TOKEN ';'
--            ..prag.. set_exec_end ;


.seq_of_stmts.:
  ..prag..
  |seq_of_stmts;


...name..:
  |   ...name.. ',' name  ;


..compilation_unit.. :
  ..prag..
  |..compilation_unit.. compilation_unit ..prag.. ;


pkg_body_or_subprg_body  : pkg_body ;



..with_cl..use_cl.... :
  |..with_cl..use_cl.... with_cl use_cls ;


use_cls :
  ..prag..
  |use_cls use_cl ..prag.. ;


...sim_n.. :
  |   ...sim_n.. ',' sim_n  ;


..or_excptn_choice.. :
  |   ..or_excptn_choice.. 'l' excptn_choice ;
```

```
..gen_prm_d.. :
  |..gen_prm_d.. gen_prm_d ;

.IN.OUT.. :
  .IN.
  |IN_TOKEN OUT_TOKEN ;

.IS_BOX_.:
  |IS_TOKEN name
  |IS_TOKEN BOX ;

PROCEDURE__ident__IS_  : subprg_spec IS_TOKEN ;


.gen_act_part. :
  |gen_act_part ;


...gen_asc.. :
  |  ...gen_asc.. ',' gen_asc ;


--.gen_fml_prmARROW.gen_act_prm :
--                        set_dec_start
--                          gen_act_prm
--                        set_dec_end
--                       |
--                        set_dec_start
--                          gen_fml_prm ARROW gen_act_prm
--                        set_dec_end
--                       ;


.gen_fml_prmARROW.gen_act_prm :
  gen_act_prm
  |gen_fml_prm ARROW gen_act_prm ;


expr_or_name_or_subprg_n_or_ent_n_or_ty_mk
          : expr ;



FOR__ty_sim_n__USE_ :
  FOR_TOKEN sim_n USE_TOKEN;


.algt_cl. :
  ..prag..
```

```
  |   ..prag.. algt_cl ..prag..  ;

..cmpon_cl.. :
  |   ..cmpon_cl.. cmpon_cl ..prag..  ;

ty_mk_rec_aggr  :  qualified_expr     ;


%%


package parser is

   procedure yyparse;

   echo : boolean := false;
   number_of_errors : natural := 0;


end parser;

with ada_tokens, ada_goto, ada_shift_reduce, ada_lex, text_io, GLOBAL;
use  ada_tokens, ada_goto, ada_shift_reduce, ada_lex, text_io, GLOBAL;
package body parser is

   procedure yyerror(s: in string := "syntax error") is
   begin
  number_of_errors := number_of_errors + 1;
  put("<<< *** ");
  put_line(s);
   end yyerror;



##%procedure_parse

end parser;
```

ADA_LEX.L

```
--/*-------------------------------------------------------------------------*/
--/* Lexical input for LEX for LALR(1) Grammar for ANSI Ada                  */
--/*                                                                         */
--/*          Herman Fischer                                                 */
--/*          Litton Data Systems                                            */
--/*          March 26, 1984                                                 */
--/*                                                                         */
--/* Accompanies Public Domain YACC format Ada grammar                       */
--/*                                                                         */
--/*                                                                         */
--/*                                                                         */
--/*                                                                         */
--/*                                                                         */
--/*                                                                         */
--/*                                                                         */
--/*-------------------------------------------------------------------------*/

%START IDENT Z

A           [aA]
B           [bB]
C           [cC]
D           [dD]
E           [eE]
F           [fF]
G           [gG]
H           [hH]
I           [iI]
J           [jJ]
K           [kK]
L           [lL]
M           [mM]
N           [nN]
O           [oO]
P           [pP]
Q           [qQ]
R           [rR]
S           [sS]
T           [tT]
U           [uU]
V           [vV]
W           [wW]
X           [xX]
Y           [yY]
Z           [zZ]

%%
```

```
{A}{B}{O}{R}{T}       {ECHO; ENTER(Z); return(ABORT_TOKEN);}
{A}{B}{S}           {ECHO; ENTER(Z); return(ABS_TOKEN);}
{A}{C}{C}{E}{P}{T}    {ECHO; ENTER(Z); return(ACCEPT_TOKEN);}
{A}{C}{C}{E}{S}{S}    {ECHO; ENTER(Z); return(ACCESS_TOKEN);}
{A}{L}{L}           {ECHO; ENTER(Z); return(ALL_TOKEN);}
{A}{N}{D}           {ECHO; ENTER(Z); return(AND_TOKEN);}
{A}{R}{R}{A}{Y}       {ECHO; ENTER(Z); return(ARRAY_TOKEN);}
{A}{T}            {ECHO; ENTER(Z); return(AT_TOKEN);}
{B}{E}{G}{I}{N}       {ECHO; ENTER(Z); return(BEGIN_TOKEN);}
{B}{O}{D}{Y}          {ECHO; ENTER(Z); return(BODY_TOKEN);}
{C}{A}{S}{E}          {ECHO; ENTER(Z); return(CASE_TOKEN);}
{C}{O}{N}{S}{T}{A}{N}{T}      {ECHO; ENTER(Z); return(CONSTANT_TOKEN);}
{D}{E}{C}{L}{A}{R}{E}  {ECHO; ENTER(Z); return(DECLARE_TOKEN);}
{D}{E}{L}{A}{Y}       {ECHO; ENTER(Z); return(DELAY_TOKEN);}
{D}{E}{L}{T}{A}       {ECHO; ENTER(Z); return(DELTA_TOKEN);}
{D}{I}{G}{I}{T}{S}    {ECHO; ENTER(Z); return(DIGITS_TOKEN);}
{D}{O}            {ECHO; ENTER(Z); return(DO_TOKEN);}
{E}{L}{S}{E}          {ECHO; ENTER(Z); return(ELSE_TOKEN);}
{E}{L}{S}{I}{F}       {ECHO; ENTER(Z); return(ELSIF_TOKEN);}
{E}{N}{D}           {ECHO; ENTER(Z); return(END_TOKEN);}
{E}{N}{T}{R}{Y}       {ECHO; ENTER(Z); return(ENTRY_TOKEN);}
{E}{X}{C}{E}{P}{T}{I}{O}{N}    {ECHO; ENTER(Z); return(EXCEPTION_TOKEN);}
{E}{X}{I}{T}          {ECHO; ENTER(Z); return(EXIT_TOKEN);}
{F}{O}{R}           {ECHO; ENTER(Z); return(FOR_TOKEN);}
{F}{U}{N}{C}{T}{I}{O}{N}      {ECHO; ENTER(Z); return(FUNCTION_TOKEN);}
{G}{E}{N}{E}{R}{I}{C}  {ECHO; ENTER(Z); return(GENERIC_TOKEN);}
{G}{O}{T}{O}          {ECHO; ENTER(Z); return(GOTO_TOKEN);}
{I}{F}            {ECHO; ENTER(Z); return(IF_TOKEN);}
{I}{N}            {ECHO; ENTER(Z); return(IN_TOKEN);}
{I}{S}            {ECHO; ENTER(Z); return(IS_TOKEN);}
{L}{I}{M}{I}{T}{E}{D}  {ECHO; ENTER(Z); return(LIMITED_TOKEN);}
{L}{O}{O}{P}          {ECHO; ENTER(Z); return(LOOP_TOKEN);}
{M}{O}{D}           {ECHO; ENTER(Z); return(MOD_TOKEN);}
{N}{E}{W}           {ECHO; ENTER(Z); return(NEW_TOKEN);}
{N}{O}{T}           {ECHO; ENTER(Z); return(NOT_TOKEN);}
{N}{U}{L}{L}          {ECHO; ENTER(Z); return(NULL_TOKEN);}
{O}{F}            {ECHO; ENTER(Z); return(OF_TOKEN);}
{O}{R}            {ECHO; ENTER(Z); return(OR_TOKEN);}
{O}{T}{H}{E}{R}{S}    {ECHO; ENTER(Z); return(OTHERS_TOKEN);}
{O}{U}{T}           {ECHO; ENTER(Z); return(OUT_TOKEN);}
{P}{A}{C}{K}{A}{G}{E}  {ECHO; ENTER(Z); return(PACKAGE_TOKEN);}
{P}{R}{A}{G}{M}{A}    {ECHO; ENTER(Z); return(PRAGMA_TOKEN);}
{P}{R}{I}{V}{A}{T}{E}  {ECHO; ENTER(Z); return(PRIVATE_TOKEN);}
{P}{R}{O}{C}{E}{D}{U}{R}{E}    {ECHO; ENTER(Z); return(PROCEDURE_TOKEN);}
{R}{A}{I}{S}{E}       {ECHO; ENTER(Z); return(RAISE_TOKEN);}
{R}{A}{N}{G}{E}       {ECHO; ENTER(Z); return(RANGE_TOKEN);}
{R}{E}{C}{O}{R}{D}    {ECHO; ENTER(Z); return(RECORD_TOKEN);}
{R}{E}{M}           {ECHO; ENTER(Z); return(REM_TOKEN);}
{R}{E}{N}{A}{M}{E}{S}  {ECHO; ENTER(Z); return(RENAMES_TOKEN);}
{R}{E}{T}{U}{R}{N}    {ECHO; ENTER(Z); return(RETURN_TOKEN);}
{R}{E}{V}{E}{R}{S}{E}  {ECHO; ENTER(Z); return(REVERSE_TOKEN);}
```

```
{S}{E}{L}{E}{C}{T}     {ECHO; ENTER(Z); return(SELECT_TOKEN);}
{S}{E}{P}{A}{R}{A}{T}{E}       {ECHO; ENTER(Z); return(SEPARATE_TOKEN);}
{S}{U}{B}{T}{Y}{P}{E}   {ECHO; ENTER(Z); return(SUBTYPE_TOKEN);}
{T}{A}{S}{K}          {ECHO; ENTER(Z); return(TASK_TOKEN);}
{T}{E}{R}{M}{I}{N}{A}{T}{E}     {ECHO; ENTER(Z); return(TERMINATE_TOKEN);}
{T}{H}{E}{N}          {ECHO; ENTER(Z); return(THEN_TOKEN);}
{T}{Y}{P}{E}          {ECHO; ENTER(Z); return(TYPE_TOKEN);}
{U}{S}{E}            {ECHO; ENTER(Z); return(USE_TOKEN);}
{W}{H}{E}{N}          {ECHO; ENTER(Z); return(WHEN_TOKEN);}
{W}{H}{I}{L}{E}       {ECHO; ENTER(Z); return(WHILE_TOKEN);}
{W}{I}{T}{H}          {ECHO; ENTER(Z); return(WITH_TOKEN);}
{X}{O}{R}            {ECHO; ENTER(Z); return(XOR_TOKEN);}
"=>"    {ECHO; ENTER(Z); return(ARROW);}
".."    {ECHO; ENTER(Z); return(DOUBLE_DOT);}
"**"    {ECHO; ENTER(Z); return(DOUBLE_STAR);}
":="    {ECHO; ENTER(Z); return(ASSIGNMENT);}
"/="    {ECHO; ENTER(Z); return(INEQUALITY);}
">="    {ECHO; ENTER(Z); return(GREATER_THAN_OR_EQUAL);}
"<="    {ECHO; ENTER(Z); return(LESS_THAN_OR_EQUAL);}
"<<"    {ECHO; ENTER(Z); return(LEFT_LABEL_BRACKET);}
">>"    {ECHO; ENTER(Z); return(RIGHT_LABEL_BRACKET);}
"<>"    {ECHO; ENTER(Z); return(BOX);}
"&"     {ECHO; ENTER(Z); return('&'); }
"("     {ECHO; ENTER(Z); return('('); }
")"     {ECHO; ENTER(IDENT); return(')'); }
"*"     {ECHO; ENTER(Z); return('*'); }
"+"     {ECHO; ENTER(Z); return('+'); }
","     {ECHO; ENTER(Z); return(','); }
"-"     {ECHO; ENTER(Z); return('-'); }
"."     {ECHO; ENTER(Z); return('.'); }
"/"     {ECHO; ENTER(Z); return('/'); }
":"     {ECHO; ENTER(Z); return(':'); }
";"     {ECHO; ENTER(Z); return(';'); }
"<"     {ECHO; ENTER(Z); return('<'); }
"="     {ECHO; ENTER(Z); return('='); }
">"     {ECHO; ENTER(Z); return('>'); }
"|"     {ECHO; ENTER(Z); return('|'); }
<IDENT>\'  {ECHO; ENTER(Z); return(''');}

[a-z_A-Z][a-z_A-Z0-9]*      {ECHO; ENTER(IDENT);return(IDENTIFIER);}
[0-9][0-9_]*([.][0-9_]+)?([Ee][-+]?[0-9_]+)?  {
              ECHO; ENTER(Z);
              return(REAL_LITERAL);}

[0-9][0-9_]*#[0-9a-fA-F_]+([.][0-9a-fA-F_]+)?#([Ee][-+]?[0-9_]+)? {
              ECHO; ENTER(Z);
              return(INTEGER_LITERAL);}

\"([^"]*(\'\")*)*\" {ECHO; ENTER(Z); return(STRING_LITERAL);}

<Z>\'([^']|\'\')\'    {ECHO; ENTER(Z); return(CHARACTER_LITERAL);}
```

```
--
-- Looking for an elsif on a line by itself
--
^[ \t]*"elsif"[ \t]*\n { ECHO;
                ENTER(Z);
                --put_line (" just found a elsif on a line by itself ");
                if GLOBAL.COUNT_CLARIFICATION
(GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_11) then
                 --put (" exec start ");
                 --put (" exec end ");
                 --new_line;
                 GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 1) := TRUE;
                 GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 2) := TRUE;
                 GLOBAL.ADD_TO_ARRAY;
                else
                 null;
                end if;
                linenum;
                return(ELSIF_TOKEN); }

--
-- Looking for an "else" on a line by itself
--
^[ \t]*"else"[ \t]*\n  { ECHO;
                ENTER(Z);
                --put_line (" just found a else on a line by itself ");
                if GLOBAL.COUNT_CLARIFICATION
(GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_10) then
                 --put (" exec start ");
                 --put (" exec end ");
                 --new_line;
                 GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 1) := TRUE;
                 GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 2) := TRUE;
                 GLOBAL.ADD_TO_ARRAY;
                else
                 null;
                end if;
                linenum;
                return(ELSE_TOKEN); }

--
-- Looking for a "then" on a line by itself
--
^[ \t]*"then"[ \t]*\n   { ECHO;
                ENTER(Z);
                --put_line (" just found a then on a line by itself ");
                if GLOBAL.COUNT_CLARIFICATION
(GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_10) then
                 --put (" exec start ");
                 --put (" exec end ");
```

```
                --new_line;
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 1) := TRUE;
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 2) := TRUE;
                GLOBAL.ADD_TO_ARRAY;
               else
                null;
               end if;
               linenum;
               return(THEN_TOKEN); }


--
-- Looking for an "others" on a line by itself
--
^[\t]*"others"[\t]*\n { ECHO;
               ENTER(Z);
               --put_line (" just found a others on a line by itself ");
               if GLOBAL.COUNT_CLARIFICATION
(GLOBAL.RECORD_FLAGS_F.PANEL10.LINE_10) then
                --put (" exec start ");
                --put (" exec end ");
                new_line;
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 1) := TRUE;
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (0), 2) := TRUE;
                GLOBAL.ADD_TO_ARRAY;
               else
                null;
               end if;
               linenum;
               return(OTHERS_TOKEN); }


--
-- Looking for a banner comment of just hyphens "---", must be longer
-- than two initial hypens, otherwise it is a empty comment.
--
^[\t]*"---""-"*[\t]*\n  { ECHO;
               --   put_line (" found a banner comment of just hyphens ");
                  GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (5), 1) := TRUE;
                  GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (5), 2) := TRUE;
                  GLOBAL.ADD_TO_ARRAY;
                  linenum; }

--
-- Checking for empty comments on a line by themselves
^[\t]*"--"[\t]*\n      { ECHO;
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (6), 1) := TRUE;
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (6), 2) := TRUE;
                GLOBAL.ADD_TO_ARRAY;
                linenum; }
```

```
-- Checking for blank lines
^[ \t]*\n          { ECHO;
                   GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (7), 1) := TRUE;
                   GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (7), 1) := TRUE;
                   GLOBAL.ADD_TO_ARRAY;
                   linenum; }

[ \t]  ECHO;          -- ignore spaces and tabs


"--*_Programmed"    { ECHO;
                   GLOBAL.SPECIAL_COMMENT := TRUE;
                   GLOBAL.CURRENT_SETTINGS.SECOND_ATTRIBUTE :=
GLOBAL.HOW_PRODUCED'val (0); }

"--*_Generated"     { ECHO;
                   GLOBAL.SPECIAL_COMMENT := TRUE;
                   GLOBAL.CURRENT_SETTINGS.SECOND_ATTRIBUTE :=
GLOBAL.HOW_PRODUCED'val (1); }

"--*_Converted"     { ECHO;
                   GLOBAL.SPECIAL_COMMENT := TRUE;
                   GLOBAL.CURRENT_SETTINGS.SECOND_ATTRIBUTE :=
GLOBAL.HOW_PRODUCED'val (2); }

"--*_Copied"        { ECHO;
                   GLOBAL.CURRENT_SETTINGS.SECOND_ATTRIBUTE :=
GLOBAL.HOW_PRODUCED'val (3);
                   GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Modified"      { ECHO;
                   GLOBAL.CURRENT_SETTINGS.SECOND_ATTRIBUTE :=
GLOBAL.HOW_PRODUCED'val (4);
                   GLOBAL.SPECIAL_COMMENT := TRUE; }

-- User will need to follow the following examples if they want to include
-- removed code in their counts
-- --*_Removed Executables => 45, Declarations => 4, Pragmas => 0
-- --*_Removed Exec => 45, Dec => 4, Prag => 0
-- --*_Removed E => 45, D => 4, P => 0

"--*_Removed ".* { ECHO;
                GLOBAL.CURRENT_SETTINGS.SECOND_ATTRIBUTE :=
GLOBAL.HOW_PRODUCED'val (5);
                GLOBAL.SPECIAL_COMMENT := TRUE;
                GLOBAL.SPEC_COMMENT_LENGTH := ada_lex_dfa.yytext'length;
               GLOBAL.SPEC_COMMENT_STRING (1 .. GLOBAL.SPEC_COMMENT_LENGTH)
:= ada_lex_dfa.yytext;
                GLOBAL.PARSE_SPECIAL_COMMENT (GLOBAL.REMOVED_NUM,
                                  GLOBAL.SPEC_COMMENT_LENGTH,
                                  GLOBAL.SPEC_COMMENT_STRING); }
```

```
"--*_New_work"      { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(0);
                GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Previous_version"  { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(1);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_COTS"          { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(2);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_GFS"           { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(3);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Annother_product"  { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(4);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_VSL_spt_library"   { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(5);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_VS_OS_or_utility"  { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(6);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_A_modified_spt_lib" { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(7);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Other_comm_lib"    { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(8);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Reuse_library"     { ECHO;
                GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE := GLOBAL.ORGIN'val
(9);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }
```

```
"--*_Other_Software_component" { ECHO;
                    GLOBAL.CURRENT_SETTINGS.THIRD_ATTRIBUTE :=
GLOBAL.ORGIN'val (10);
                    GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Part_of_product"  { ECHO;
                GLOBAL.CURRENT_SETTINGS.FOURTH_ATTRIBUTE :=
GLOBAL.USAGE'val (0);
                GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_External_to_product" { ECHO;
                 GLOBAL.CURRENT_SETTINGS.FOURTH_ATTRIBUTE :=
GLOBAL.USAGE'val (1);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

-- To include an estimated or planned value for executable, declarations, and or pragmas, the
-- user will need to follow the following examples
-- --*_Estimated Executables => 2245, Declarations => 400, Pragmas => 14
-- --*_Estimated Exec => 2245, Dec => 400, Prag => 14
-- --*_Estimated E => 2245, D => 400, P => 14

"--*_Estimated_or_planned ".* { ECHO;
                    GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE :=
GLOBAL.DEVELOPMENT_STATUS'val (0);
                    GLOBAL.SPECIAL_COMMENT := TRUE;
                    GLOBAL.SPEC_COMMENT_LENGTH := ada_lex_dfa.yytext'length;
                    GLOBAL.SPEC_COMMENT_STRING (1 ..
GLOBAL.SPEC_COMMENT_LENGTH) := ada_lex_dfa.yytext;
                    GLOBAL.PARSE_SPECIAL_COMMENT (GLOBAL.ESTIMATED_NUM,
                                GLOBAL.SPEC_COMMENT_LENGTH,
                                GLOBAL.SPEC_COMMENT_STRING); }

"--*_Designed"  { ECHO;
            GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE :=
GLOBAL.DEVELOPMENT_STATUS'val (1);
            GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Coded"     { ECHO;
            GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE :=
GLOBAL.DEVELOPMENT_STATUS'val (2);
            GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Unit_tests_completed" { ECHO;
                 GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE :=
GLOBAL.DEVELOPMENT_STATUS'val (3);
                 GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_Integrated_into_components" { ECHO;
                    GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE :=
GLOBAL.DEVELOPMENT_STATUS'val (4);
                    GLOBAL.SPECIAL_COMMENT := TRUE; }
```

```
"--*_Test_readiness_rev_completed" { ECHO;
                        GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE:=
GLOBAL.DEVELOPMENT_STATUS'val (5);
                        GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_CSCI_completed"        { ECHO;
                      GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE :=
GLOBAL.DEVELOPMENT_STATUS'val (6);
                      GLOBAL.SPECIAL_COMMENT := TRUE; }

"--*_System_tests_completed"  { ECHO;
                      GLOBAL.CURRENT_SETTINGS.FIFTH_ATTRIBUTE :=
GLOBAL.DEVELOPMENT_STATUS'val (7);
                      GLOBAL.SPECIAL_COMMENT := TRUE; }


--
-- Checking for comments on their own line
--
^[ \t]*"-- ".*\n      { ECHO;
                GLOBAL.THIRD_CHAR := ada_lex_dfa.yytext (3);
                GLOBAL.SPEC_COMMENT_LENGTH := ada_lex_dfa.yytext'length;
                GLOBAL.SPEC_COMMENT_STRING (1 ..
GLOBAL.SPEC_COMMENT_LENGTH):= ada_lex_dfa.yytext;
                GLOBAL.DETERMINE_TYPE_COMMENT
(GLOBAL.SPEC_COMMENT_LENGTH,
                                    GLOBAL.THIRD_CHAR,
                                    GLOBAL.SPEC_COMMENT_STRING);
                GLOBAL.ADD_TO_ARRAY;
                linenum; }

--
-- Looking for a comment on a line with source code.
-- Conditions:
-- Actions:
--
"--".*          { ECHO;
               if GLOBAL.EXECLEVEL  > 0  or
                  GLOBAL.DECLEVEL   > 0  or
                  GLOBAL.PRAGMALEVEL > 0    then
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (4), 1) := TRUE;
                GLOBAL.FLAGS_ARRAY (GLOBAL.STMT_TYPE'val (4), 2) := TRUE;
               else
                null;
               end if; }


.          { ECHO;
             text_io.put_line("?? lexical error" & ada_lex_dfa.yytext & "??");
           num_errors := num_errors + 1; }
```

```
[\n]     { ECHO;
           GLOBAL.ADD_TO_ARRAY;
           linenum; }

%%

with TEXT_IO,
   ada_tokens,
   GLOBAL,
   ada_lex_dfa,
   TAE;
use  ada_tokens;

package ada_lex is

 lines              : positive := 1;
 num_errors           : natural := 0;

 procedure DECREASE_DECLEVEL;

 procedure linenum;

 function yylex return token;

end ada_lex;

package body ada_lex is


 --
 --
 procedure DECREASE_DECLEVEL is
 begin
  GLOBAL.DECLEVEL := 0;
 end DECREASE_DECLEVEL;


 --
 --
 procedure linenum is
 begin
  text_io.put(integer'image(lines) & ":");
  lines := lines + 1;
 end linenum;

##

end ada_lex;
```

GLOBAL_S.A

```
-- *** TAE Plus Code Generator version V5.1
-- *** File:       global_s.a
-- *** Generated:  Apr 15 10:49:42 1993
-- ******************************************************************************
-- *
-- *   Global                    -- Package SPEC
-- *
-- ******************************************************************************

with X_Windows;
with Text_IO;
with TAE;
use  TAE;
package Global is

--| PURPOSE:
--| This package is automatically "with"ed in to each panel package body.
--| You can insert global variables here.
--|
--| INITIALIZATION EXCEPTIONS: (none)
--|
--| NOTES: (none)
--|
--| REGENERATED:
--| This file is generated only once.
--|
--| CHANGE LOG:
--| 15-Apr-93   TAE    Generated

--*_Programmed

 -- (+)  begin added code
 --

 type MY_VALUE is array (1..1) of String (1..TAE.Tae_Taeconf.STRINGSIZE);

 SPECIAL_COMMENT,
 COMMENT_FLAG          : boolean := false;

 SPEC_COMMENT_LENGTH,
 EXECLEVEL,
 DECLEVEL,
 PRAGMALEVEL           : integer := 0;

 REMOVED_NUM           : integer := -1;
 ESTIMATED_NUM         : integer := 1;

 THIRD_CHAR            : character := ' ';
```

```
SPEC_COMMENT_STRING    : string (1 .. 1024) := (others => ' ');

OUT_FILE_TYPE    : text_io.file_type;
F                : text_io.file_type;
FILE_LIST_NAME   : string (1 .. 1024) := (others => ' ');

type STMT_TYPE is (EXECUTABLE, DECLARATIONS, COMPILER_DIRECTIVES,
            CMTS_ON_OWN_LINE, CMTS_WITH_SRC_CODE,
            BANNERS_NON_BLANK_SPACERS, BLANK_EMPTY_CMTS,
            BLANK_LINES);

type HOW_PRODUCED is (PROGRAMMED, GENERATED, COVERETED,
              COPIED, MODIFIED, REMOVED);

type ORGIN is (NEW_WORK, PREVIOUS_VERSION, COTS, GFS, ANNOTHER_PRODUCT,
          VENDOR_SUPPLIED_SPT_LIB, VENDOR_SUPPLIED_OS,
          LOCAL_SUPPLIED_LIB, COMMERCIAL_LIB, REUSE_LIB,
          OTHER_COMPONENT_LIB);

type USAGE is (PRIMARY_PRODUCT, EXTERNAL);

type DEVELOPMENT_STATUS is (ESTIMATED, DESIGNED, CODED, UNIT_TEST_DONE,
                  INTEGRATED, TEST_READINESS_REVIEW,
                  CSCI_COMPLETED, SYSTEM_TESTS_COMPLETED);

type COUNT_ARRAY_TYPE is array (STMT_TYPE, HOW_PRODUCED,
                     ORGIN, USAGE, DEVELOPMENT_STATUS) of natural;

type FLAGS_TYPE_ARRAY is array (STMT_TYPE, 1..2) of boolean;

type PRIORITY_TYPE_ARRAY is array (1..8) of STMT_TYPE;

type ORDER_OF_PRECEDENCE  is range 1..8;

type CURRENT_SETTINGS_TYPE is
 record
  FIRST_ATTRIBUTE      : STMT_TYPE        := EXECUTABLE;
  SECOND_ATTRIBUTE     : HOW_PRODUCED     := PROGRAMMED;
  THIRD_ATTRIBUTE      : ORGIN            := NEW_WORK;
  FOURTH_ATTRIBUTE     : USAGE            := PRIMARY_PRODUCT;
  FIFTH_ATTRIBUTE      : DEVELOPMENT_STATUS := SYSTEM_TESTS_COMPLETED;
 end record;

type STMT_TOTALS_TYPE is
 record
  EXEC_TOTAL,
  DEC_TOTAL,
  PRAGMA_TOTAL,
  CMTS_ON_OWN_TOTAL,
  CMTS_W_SRC_TOTAL,
```

```
    BANNER_CMTS_TOTAL,
    EMPTY_CMTS_TOTAL,
    BLANK_LINES_TOTAL     : natural := 0;
  end record;

type HOW_PRODUCED_TYPE is
  record
    PROGRAMMED_TOTAL,
    GENERATED_TOTAL,
    CONVERTED_TOTAL,
    COPIED_TOTAL,
    MODIFIED_TOTAL,
    REMOVED_TOTAL         : natural := 0;
  end record;

type ORGIN_TYPE is
  record
    NEW_WORK_TOTAL,
    PREVIOUS_VERSION_TOTAL,
    COTS_TOTAL,
    GFS_TOTAL,
    ANNOTHER_PRODUCT_TOTAL,
    VS_SPT_LIB_TOTAL,
    VS_SPT_OS_TOTAL,
    LOCAL_SUPPLIED_LIB_TOTAL,
    COMMERCIAL_LIB_TOTAL,
    REUSE_LIB_TOTAL,
    OTHER_COMPONENT_TOTAL     : natural := 0;
  end record;

type USAGE_TYPE is
  record
    PRIMARY_PRODUCT_TOTAL,
    EXTERNAL_TOTAL          : natural := 0;
  end record;

type DEVELOPMENT_STATUS_TYPE is
  record
    ESTIMATED_TOTAL,
    DESIGNED_TOTAL,
    CODED_TOTAL,
    UNIT_TEST_DONE_TOTAL,
    INTEGRATED_TOTAL,
    TEST_READINESS_REVIEW_TOTAL,
    CSCI_COMPLETED_TOTAL,
    SYSTEM_TEST_TOTAL          : natural := 0;
  end record;

type COUNT_TOTALS_TYPE is
  record
    STMT_NUMS         : STMT_TOTALS_TYPE;
```

```
   PRODUCED_NUMS     : HOW_PRODUCED_TYPE;
   ORGIN_NUMS        : ORGIN_TYPE;
   USAGE_NUMS        : USAGE_TYPE;
   DEVELOPED_NUMS    : DEVELOPMENT_STATUS_TYPE;
  end record;

type panel_2 is
  record
   report_a          : boolean := true;
   report_b          : boolean := false;
   report_c          : boolean := false;
   report_d          : boolean := false;
   report_e          : boolean := false;
   report_f          : boolean := false;
   next_scrn         : boolean := false;
   out_file_name     : my_value;
   in_file_name      : my_value;
   requestor         : my_value;
   report_heading    : my_value;
  end record;

type panel_3 is
  record
   line_1            : boolean := true;
   line_3            : boolean := true;
   line_4            : boolean := true;
   line_6            : boolean := false;
   line_7            : boolean := false;
   line_8            : boolean := false;
   line_9            : boolean := false;
   line_10           : boolean := false;
   line_1_int        : TAE.TAEINT := 1;
   line_3_int        : TAE.TAEINT := 2;
   line_4_int        : TAE.TAEINT := 3;
   line_6_int        : TAE.TAEINT := 4;
   line_7_int        : TAE.TAEINT := 5;
   line_8_int        : TAE.TAEINT := 6;
   line_9_int        : TAE.TAEINT := 7;
   line_10_int       : TAE.TAEINT := 8;
   def_data_array    : boolean := false;
  end record;


type panel_4 is
  record
   line_1            : boolean := true;
   line_2            : boolean := true;
   line_3            : boolean := true;
   line_4            : boolean := true;
   line_5            : boolean := true;
   line_6            : boolean := false;
```

```
   def_data_array     :  boolean := false;
  end record;


type panel_5 is
  record
    line_1           :  boolean := true;
    line_3           :  boolean := true;
    line_4           :  boolean := true;
    line_5           :  boolean := true;
    line_6           :  boolean := true;
    line_7           :  boolean := false;
    line_8           :  boolean := false;
    line_9           :  boolean := true;
    line_10          :  boolean := true;
    line_11          :  boolean := true;
    line_12          :  boolean := true;
    def_data_array     :  boolean := false;
  end record;


type panel_6 is
  record
    line_1           :  boolean := true;
    line_2           :  boolean := false;
    DEL_OPTION           :  MY_VALUE;
    def_data_array     :  boolean := false;
  end record;


type panel_9 is
  record
    line_1           :  boolean := false;
    line_2           :  boolean := false;
    line_3           :  boolean := false;
    line_4           :  boolean := false;
    line_5           :  boolean := false;
    line_6           :  boolean := false;
    line_7           :  boolean := false;
    line_8           :  boolean := true;
    def_data_array     :  boolean := false;
  end record;


type panel_10 is
  record
    line_1           :  boolean := true;
    line_2           :  boolean := true;
    line_3           :  boolean := true;
    line_4           :  boolean := true;
    line_5           :  boolean := true;
```

```
    line_6          :  boolean := true;
    line_7          :  boolean := true;
    line_8          :  boolean := true;
    line_9          :  boolean := true;
    line_10         :  boolean := true;
    line_11         :  boolean := true;
    line_12         :  boolean := true;
    line_13         :  boolean := true;
    line_1_int      :  TAE.TAEINT := 1;
    line_2_int      :  TAE.TAEINT := 1;
    line_3_int      :  TAE.TAEINT := 3;
    line_4_int      :  TAE.TAEINT := 1;
    line_5_int      :  TAE.TAEINT := 3;
    line_6_int      :  TAE.TAEINT := 1;
    line_7_int      :  TAE.TAEINT := 1;
    line_8_int      :  TAE.TAEINT := 1;
    line_9_int      :  TAE.TAEINT := 3;
    line_10_int     :  TAE.TAEINT := 1;
    line_11_int     :  TAE.TAEINT := 1;
    line_12_int     :  TAE.TAEINT := 3;
    line_13_int     :  TAE.TAEINT := 1;
    def_data_array  :  boolean := false;
  end record;

type panel_11 is
  record
    line_1          :  boolean := true;
    line_2          :  boolean := true;
    line_3          :  boolean := true;
    line_4          :  boolean := true;
    line_5          :  boolean := true;
    line_6          :  boolean := true;
    line_1_int      :  TAE.TAEINT := 3;
    line_2_int      :  TAE.TAEINT := 1;
    line_3_int      :  TAE.TAEINT := 3;
    line_4_int      :  TAE.TAEINT := 1;
    line_5_int      :  TAE.TAEINT := 3;
    line_6_int      :  TAE.TAEINT := 4;
  end record;

type flags is
  record
    panel2          : panel_2;
    panel3          : panel_3;
    panel4          : panel_4;
    panel5          : panel_5;
    panel6          : panel_6;
    panel9          : panel_9;
    panel10         : panel_10;
    panel11         : panel_11;
  end record;
```

```
record_flags          : flags;
record_flags_A        : flags;
record_flags_B        : flags;
record_flags_C        : flags;
record_flags_D        : flags;
record_flags_E        : flags;
record_flags_F        : flags;

COUNT_ARRAY_A     : COUNT_ARRAY_TYPE := (others => (others =>
                     (others => (others =>
                      (others => 0)))));

COUNT_ARRAY_B     : COUNT_ARRAY_TYPE := (others => (others =>
                     (others => (others =>
                      (others => 0)))));

COUNT_ARRAY_C     : COUNT_ARRAY_TYPE := (others => (others =>
                     (others => (others =>
                      (others => 0)))));

COUNT_ARRAY_D     : COUNT_ARRAY_TYPE := (others => (others =>
                     (others => (others =>
                      (others => 0)))));

COUNT_ARRAY_E     : COUNT_ARRAY_TYPE := (others => (others =>
                     (others => (others =>
                      (others => 0)))));

COUNT_ARRAY_F     : COUNT_ARRAY_TYPE := (others => (others =>
                     (others => (others =>
                      (others => 0)))));

FLAGS_ARRAY       : FLAGS_TYPE_ARRAY;
PRIORITY_ARRAY_A_E : PRIORITY_TYPE_ARRAY;
PRIORITY_ARRAY_F   : PRIORITY_TYPE_ARRAY;

CURRENT_SETTINGS   : CURRENT_SETTINGS_TYPE;

COUNT_TOTALS      : COUNT_TOTALS_TYPE;
COUNT_TOTALS_A    : COUNT_TOTALS_TYPE;
COUNT_TOTALS_B    : COUNT_TOTALS_TYPE;
COUNT_TOTALS_C    : COUNT_TOTALS_TYPE;
COUNT_TOTALS_D    : COUNT_TOTALS_TYPE;
COUNT_TOTALS_E    : COUNT_TOTALS_TYPE;
COUNT_TOTALS_F    : COUNT_TOTALS_TYPE;

TOTAL_COUNTED_A,
TOTAL_COUNTED_B,
TOTAL_COUNTED_C,
TOTAL_COUNTED_D,
```

```
  TOTAL_COUNTED_E,
  TOTAL_COUNTED_F    : natural := 0;



  --
  -- (-)  end added code

--*_Generated

  package Taefloat_IO is new Text_IO.Float_IO (TAE.Taefloat);

  Default_Display_Id : X_Windows.Display;



--  procedure CHECK_FLAG_SETTINGS;


  -- ..........................................................
  -- .
  -- .   Application_Done              -- Subprogram SPEC
  -- .
  -- ..........................................................

  function Application_Done
    return Boolean;

  --| PURPOSE:
  --| This function returns true if a "quit" event handler has called
  --| Set_Application_Done, otherwise it returns false.
  --|
  --| EXCEPTIONS: (none)
  --|
  --| NOTES: (none)




  -- ..........................................................  ...
  -- .
  -- .   Set_Application_Done          -- Subprogram SPEC
  -- .
  -- ..........................................................

  procedure Set_Application_Done;

  --| PURPOSE:
  --| This procedure can be used by an event handler, typically a "quit"
```

```
--| button, to signal the end of the application.
--|
--| EXCEPTIONS: (none)
--|
--| NOTES: (none)


-- .....
-- .
-- .   Switch_flag              -- Subprogram SPEC
-- .
-- .....


function  Switch_Flag (FLAG_IN : in boolean) return boolean;

--| PURPOSE:
--| This procedure will be used when the user changes the default settings
--| for the custom format report.
--|
--| EXCEPTIONS:  (none)
--|
--| NOTES:  (none)

--
--
function CHECK_REPORT_A_E return boolean;


--
--
function CHECK_REPORT_F return boolean;

--
--
function COUNT_CLARIFICATION (BOOLEAN_IN : in BOOLEAN) return BOOLEAN;
--|
--|
--|
--|

--
--
procedure OPEN_OUT_FILE;

--
--
procedure CLOSE_OUT_FILE;

--
```

```
  --
  function FIND_LENGTH (FILELIST : in GLOBAL.MY_VALUE) return integer;

  --
  --
  procedure ADD_TO_ARRAY;

  --
  --
  procedure COUNT_LINE (IN_RECORD  : in    CURRENT_SETTINGS_TYPE;
                        ARRAY_TYPE : in out COUNT_ARRAY_TYPE;
                        ADD_NUMBER : in     natural := 1);

  --
  --
  procedure DETERMINE_WHICH_ARRAY (IN_RECORD  : in
CURRENT_SETTINGS_TYPE;
                                   ADD_NUMBER : in     natural := 1 );

  --
  --
  procedure PARSE_SPECIAL_COMMENT (IN_NUM    : in integer;
                                   IN_LENGTH : in integer;
                                   IN_STRING : in string);

  --
  --
  procedure DETERMINE_TYPE_COMMENT (IN_BANNER_LENGTH  : in out integer;
                                    IN_BANNER_CHAR    : in out character;
                                    IN_BANNER_STRING  : in out STRING);



end Global;
```

GLOBAL_B.A

```
-- *** TAE Plus Code Generator version V5.1
-- *** File:       global_b.a
-- *** Generated:   Apr 15 10:49:42 1993
-- ******************************************************************************
-- *
-- *   Global                      -- Package BODY
-- *
-- ******************************************************************************

with TEXT_IO;
use  TEXT_IO;
package body Global is

--| NOTES: (none)
--|
--| REGENERATED:
--| This file is generated only once.
--|
--| CHANGE LOG:
--| 15-Apr-93   TAE    Generated

--*_Programmed

 package TAE_INTEGER_IN_OUT is new integer_io (TAE.TAEINT);
 use TAE_INTEGER_IN_OUT;

 package INTEGER_IN_OUT is new integer_io (integer);
 use INTEGER_IN_OUT;

 package ENUMERATION_IN_OUT is new ENUMERATION_IO (STMT_TYPE);
 use ENUMERATION_IN_OUT;

--*_Generated

 Is_Application_Done : Boolean := FALSE;

 -- ..........................................................
 -- .
 -- .   Application_Done             -- Subprogram BODY
 -- .
 -- ..........................................................

 function Application_Done
   return Boolean is

 --| NOTES: (none)
```

```
begin -- Application_Done

 return Is_Application_Done;

end Application_Done;


-- .....................................................................
-- .
-- .   Set_Application_Done          -- Subprogram BODY
-- .
-- .....................................................................

procedure Set_Application_Done is

--| NOTES: (none)

begin -- Set_Application_Done

 Is_Application_Done := TRUE;

end Set_Application_Done;


--*_Programmed

-- .....................................................................
-- .
-- .   Switch_flag              -- Subprogram SPEC
-- .
-- .....................................................................



function Switch_Flag (FLAG_IN : in boolean) return boolean is

--| PURPOSE:
--| This procedure will be used when the user changes the default settings
--| for the custom format report.
--|
--| EXCEPTIONS:  (none)
--|
--| NOTES:  (none)

 TEMP_FLAG     : boolean;

begin
 if FLAG_IN then
  TEMP_FLAG := false;
 else
  TEMP_FLAG := true;
```

```
 end if;

 return TEMP_FLAG;

end Switch_Flag;


--
--
function CHECK_REPORT_A_E return boolean is

 BOOLEAN_FLAG  :  boolean := FALSE;

begin

 if  RECORD_FLAGS.PANEL2.REPORT_A or
     RECORD_FLAGS.PANEL2.REPORT_B or
     RECORD_FLAGS.PANEL2.REPORT_C or
     RECORD_FLAGS.PANEL2.REPORT_D or
     RECORD_FLAGS.PANEL2.REPORT_E    then
   BOOLEAN_FLAG := TRUE;
 end if;

 return BOOLEAN_FLAG;

end CHECK_REPORT_A_E;

--
--
function CHECK_REPORT_F return boolean is

 BOOLEAN_FLAG  :  boolean := FALSE;

begin

 if  RECORD_FLAGS.PANEL2.REPORT_F then
   BOOLEAN_FLAG := TRUE;
 end if;

 return BOOLEAN_FLAG;

end CHECK_REPORT_F;


--
--
function COUNT_STMT_TYPE (S            : in STMT_TYPE;
                          IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT   :  integer  := 0;
```

```
begin

 for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
  for O in ORGIN'FIRST .. ORGIN'LAST  loop
   for U in USAGE'FIRST .. USAGE'LAST loop
    for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
     TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
    end loop;
   end loop;
  end loop;
 end loop;

 return TEMP_COUNT;

end COUNT_STMT_TYPE;


--
--
function COUNT_HOW_PRODUCED (H            : in HOW_PRODUCED;
                  IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT   :  integer  := 0;

begin

 for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
  for O in ORGIN'FIRST .. ORGIN'LAST loop
   for U in USAGE'FIRST .. USAGE'LAST loop
    for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
     TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
    end loop;
   end loop;
  end loop;
 end loop;

 return TEMP_COUNT;

end COUNT_HOW_PRODUCED;


function COUNT_ORGIN (O            : in ORGIN;
              IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT   :  integer  := 0;

begin

 for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
  for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
   for U in USAGE'FIRST .. USAGE'LAST loop
```

```
      for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
       TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
      end loop;
     end loop;
    end loop;
   end loop;

   return TEMP_COUNT;

end COUNT_ORGIN;


function COUNT_USAGE (U           : in USAGE;
                      IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

   TEMP_COUNT  :  integer := 0;

begin

   for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
    for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
     for O in ORGIN'FIRST .. ORGIN'LAST loop
      for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
       TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
      end loop;
     end loop;
    end loop;
   end loop;

   return TEMP_COUNT;

end COUNT_USAGE;


function COUNT_DEVELOPMENT_STATUS (D           : in DEVELOPMENT_STATUS;
                                   IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

   TEMP_COUNT  :  integer := 0;

begin

   for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
    for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
     for O in ORGIN'FIRST .. ORGIN'LAST loop
      for U in USAGE'FIRST .. USAGE'LAST loop
       TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
      end loop;
     end loop;
    end loop;
   end loop;
```

```
  return TEMP_COUNT;

 end COUNT_DEVELOPMENT_STATUS;

 --
 --
 procedure COUNT_ATTRIBUTE_ONE (IN_RECORD_FLAGS : in    FLAGS;
                    IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                    IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

  if IN_RECORD_FLAGS.PANEL3.line_1 then
   IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL := COUNT_STMT_TYPE
(STMT_TYPE'val (0), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_3 then
   IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL := COUNT_STMT_TYPE
(STMT_TYPE'val (1), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_4 then
   IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL := COUNT_STMT_TYPE
(STMT_TYPE'val (2), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_6 then
   IN_COUNT_TOTALS.STMT_NUMS.CMTS_ON_OWN_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (3), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_7 then
   IN_COUNT_TOTALS.STMT_NUMS.CMTS_W_SRC_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (4), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_8 then
   IN_COUNT_TOTALS.STMT_NUMS.BANNER_CMTS_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (5), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_9 then
   IN_COUNT_TOTALS.STMT_NUMS.EMPTY_CMTS_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (6), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_10 then
   IN_COUNT_TOTALS.STMT_NUMS.BLANK_LINES_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (7), IN_ARRAY);
  end if;

 end COUNT_ATTRIBUTE_ONE;

 --
 --
 procedure COUNT_ATTRIBUTE_TWO (IN_RECORD_FLAGS : in    FLAGS;
                    IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
```

```
                    IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

  if IN_RECORD_FLAGS.PANEL4.line_1 then
   IN_COUNT_TOTALS.PRODUCED_NUMS.PROGRAMMED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (0), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL4.line_2 then
   IN_COUNT_TOTALS.PRODUCED_NUMS.GENERATED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (1), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL4.line_3 then
   IN_COUNT_TOTALS.PRODUCED_NUMS.CONVERTED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (2), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL4.line_4 then
   IN_COUNT_TOTALS.PRODUCED_NUMS.COPIED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (3), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL4.line_5 then
   IN_COUNT_TOTALS.PRODUCED_NUMS.MODIFIED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (4), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL4.line_6 then
   IN_COUNT_TOTALS.PRODUCED_NUMS.REMOVED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (5), IN_ARRAY);
  end if;

 end COUNT_ATTRIBUTE_TWO;



 procedure COUNT_ATTRIBUTE_THREE (IN_RECORD_FLAGS : in    FLAGS;
                    IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                    IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

  if IN_RECORD_FLAGS.PANEL5.line_1 then
   IN_COUNT_TOTALS.ORGIN_NUMS.NEW_WORK_TOTAL := COUNT_ORGIN
(ORGIN'val (0), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL5.line_3 then
   IN_COUNT_TOTALS.ORGIN_NUMS.PREVIOUS_VERSION_TOTAL :=
          COUNT_ORGIN (ORGIN'val (1), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL5.line_4 then
   IN_COUNT_TOTALS.ORGIN_NUMS.COTS_TOTAL := COUNT_ORGIN (ORGIN'val (2),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL5.line_5 then
```

```
    IN_COUNT_TOTALS.ORGIN_NUMS.GFS_TOTAL := COUNT_ORGIN (ORGIN'val (3),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_6 then
    IN_COUNT_TOTALS.ORGIN_NUMS.ANNOTHER_PRODUCT_TOTAL :=
            COUNT_ORGIN (ORGIN'val (4), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_7 then
    IN_COUNT_TOTALS.ORGIN_NUMS.VS_SPT_LIB_TOTAL := COUNT_ORGIN
(ORGIN'val (5), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_8 then
    IN_COUNT_TOTALS.ORGIN_NUMS.VS_SPT_OS_TOTAL := COUNT_ORGIN
(ORGIN'val (6), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_9 then
    IN_COUNT_TOTALS.ORGIN_NUMS.LOCAL_SUPPLIED_LIB_TOTAL :=
            COUNT_ORGIN (ORGIN'val (7), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_10 then
    IN_COUNT_TOTALS.ORGIN_NUMS.COMMERCIAL_LIB_TOTAL :=
            COUNT_ORGIN (ORGIN'val (8), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_11 then
    IN_COUNT_TOTALS.ORGIN_NUMS.REUSE_LIB_TOTAL := COUNT_ORGIN
(ORGIN'val (9), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_12 then
    IN_COUNT_TOTALS.ORGIN_NUMS.OTHER_COMPONENT_TOTAL :=
            COUNT_ORGIN (ORGIN'val (10), IN_ARRAY);
   end if;


 end COUNT_ATTRIBUTE_THREE;

 --
 --
 procedure COUNT_ATTRIBUTE_FOUR (IN_RECORD_FLAGS : in    FLAGS;
                    IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                    IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

   if IN_RECORD_FLAGS.PANEL6.line_1 then
    IN_COUNT_TOTALS.USAGE_NUMS.PRIMARY_PRODUCT_TOTAL :=
            COUNT_USAGE (USAGE'val (0), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL6.line_2 then
    IN_COUNT_TOTALS.USAGE_NUMS.EXTERNAL_TOTAL := COUNT_USAGE
(USAGE'val (1), IN_ARRAY);
   end if;
```

```
 end COUNT_ATTRIBUTE_FOUR;

 --
 --
 procedure COUNT_ATTRIBUTE_FIVE (IN_RECORD_FLAGS : in    FLAGS;
                      IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                      IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

  if IN_RECORD_FLAGS.PANEL9.line_1 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.ESTIMATED_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (0),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL9.line_2 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.DESIGNED_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (1),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL9.line_3 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.CODED_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (2),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL9.line_4 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.UNIT_TEST_DONE_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (3),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL9.line_5 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.INTEGRATED_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (4),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL9.line_6 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.TEST_READINESS_REVIEW_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (5),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL9.line_7 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.CSCI_COMPLETED_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (6),
IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL9.line_8 then
   IN_COUNT_TOTALS.DEVELOPED_NUMS.SYSTEM_TEST_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (7),
IN_ARRAY);
  end if;

 end COUNT_ATTRIBUTE_FIVE;
```

```
  --
  --
  function COUNT_CLARIFICATION (BOOLEAN_IN : in BOOLEAN) return boolean is

    TEMP    :  boolean := FALSE;

  begin

    if BOOLEAN_IN then
      TEMP := TRUE;
    else
      TEMP := FALSE;
    end if;

    return TEMP;

  end COUNT_CLARIFICATION;


--
--
function FIND_LENGTH (FILELIST : in GLOBAL.MY_VALUE) return integer is

  TEMP_NUMBER    :  integer := 0;
  TEMP_CHAR      :  character := ' ';

begin

  for I in FILELIST'range loop
    for J in 1 .. 1024 loop
      if  FILELIST (I)(J) /= TEMP_CHAR then
        TEMP_NUMBER := TEMP_NUMBER + 1;
      else
        exit;
      end if;
    end loop;
  end loop;

  return TEMP_NUMBER;

end FIND_LENGTH;


--
--
procedure OPEN_OUT_FILE is

  OUT_FILE_NAME    :  GLOBAL.MY_VALUE :=
GLOBAL.RECORD_FLAGS.PANEL2.OUT_FILE_NAME;
  LENGTH           :  integer;
```

```
begin

  LENGTH := FIND_LENGTH (OUT_FILE_NAME);

  FILE_LIST_NAME(1..LENGTH) := OUT_FILE_NAME (1) (1..LENGTH);

  create (OUT_FILE_TYPE, out_file, FILE_LIST_NAME (1..LENGTH));

 end OPEN_OUT_FILE;


 --
 --
 procedure CLOSE_OUT_FILE is

  OUT_FILE_NAME    : GLOBAL.MY_VALUE :=
GLOBAL.RECORD_FLAGS.PANEL2.OUT_FILE_NAME;
  LENGTH          : integer;

 begin

  LENGTH := FIND_LENGTH (OUT_FILE_NAME);
  FILE_LIST_NAME(1..LENGTH) := OUT_FILE_NAME (1) (1..LENGTH);
  close (OUT_FILE_TYPE);

 end CLOSE_OUT_FILE;


 --
 --
 procedure DETERMINE_TYPE_COMMENT (IN_BANNER_LENGTH  : in out integer;
                        IN_BANNER_CHAR    : in out character;
                        IN_BANNER_STRING  : in out STRING) is

  -- Function to determine if a particular comment is
  -- a banner comment.
  function BANNER_FOUND (BANNER_LENGTH  : in integer;
                BANNER_CHAR    : in character;
                BANNER_STRING  : in STRING) return boolean is

   FIRST        : integer := 1;
   BANNER       : boolean := FALSE;
   BLANK_SPACE  : character := ' ';
   HYPHEN       : character := '-';
   BANNER_CHARS  : string (1..4) := (others => ' ');
   COUNT_LOOP    : integer := 0;
   REPEAT_CHARS  : boolean := FALSE;

  begin
```

```
    for I in BANNER_STRING'first .. BANNER_STRING'last - 1 loop

      if  BANNER_STRING (I) = '-'  and
          BANNER_STRING (I+1) = '-'    then
        BANNER_CHARS := BANNER_STRING (I+2..I+5);

        for J in BANNER_STRING'first+I+2 .. Banner_length - 4  loop

          if  BANNER_STRING (J) = BANNER_CHARS (1)  or
              BANNER_STRING (J) = BANNER_CHARS (2)  or
              BANNER_STRING (J) = BANNER_CHARS (3)  or
              BANNER_STRING (J) = BANNER_CHARS (4)  or
              BANNER_STRING (J) = BLANK_SPACE        then
            COUNT_LOOP := COUNT_LOOP + 1;
            if COUNT_LOOP > 4 then
              BANNER := TRUE;
            end if;
          else
            BANNER := FALSE;
            exit;
          end if;
          if count_loop < banner_length - 1 then
            null;
          else
            exit;
          end if;
        end loop;

        exit;
      end if;

    end loop;

    return BANNER;

  end BANNER_FOUND;


 begin

  -- Checking for banner comments.
  -- CONDITIONS:
  -- Banner characters must be non-blank;
  -- Banner characters must be either the third character from the
  -- left, or blank charactor.
  -- ACTION:
  -- Set the start and stop flags to true for Banner comments.
  if BANNER_FOUND (IN_BANNER_LENGTH, IN_BANNER_CHAR,
IN_BANNER_STRING) then
      FLAGS_ARRAY (STMT_TYPE'val (5), 1) := TRUE;
      FLAGS_ARRAY (STMT_TYPE'val (5), 2) := TRUE;
```

```
  -- Checking for comments on own line.
  -- Conditions:
  -- Start flags for Executable, Declaration, and or Pragma must not be
  -- set to true.
  -- ACTIONS:
  -- Set start and stop flags to true for Comments on own line.
  else
   FLAGS_ARRAY (STMT_TYPE'val (3), 1) := TRUE;
   FLAGS_ARRAY (STMT_TYPE'val (3), 2) := TRUE;
  end if;

 end DETERMINE_TYPE_COMMENT;


 --
 --
 function CHECKED_OKAY_A (IN_CURRENT_SETTINGS : CURRENT_SETTINGS_TYPE)
return boolean is

  OKAY   :  boolean := TRUE;

 begin

  case IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE is
   when STMT_TYPE'VAL (3) | STMT_TYPE'VAL (4) | STMT_TYPE'VAL (5) |
      STMT_TYPE'VAL (6) | STMT_TYPE'VAL (7) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.SECOND_ATTRIBUTE is
   when HOW_PRODUCED'VAL (5) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.THIRD_ATTRIBUTE is
   when ORGIN'VAL (5) | ORGIN'VAL (6) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.FOURTH_ATTRIBUTE is
   when USAGE'VAL (1) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;
```

```
  case IN_CURRENT_SETTINGS.FIFTH_ATTRIBUTE is
   when DEVELOPMENT_STATUS'VAL (0) | DEVELOPMENT_STATUS'VAL (1) |
      DEVELOPMENT_STATUS'VAL (2) | DEVELOPMENT_STATUS'VAL (3) |
      DEVELOPMENT_STATUS'VAL (4) | DEVELOPMENT_STATUS'VAL (5) |
      DEVELOPMENT_STATUS'VAL (6) =>
     OKAY := FALSE;
   when others =>
    null;
  end case;

  return OKAY;

 end CHECKED_OKAY_A;


 --
 --
 function CHECKED_OKAY_B (IN_CURRENT_SETTINGS : CURRENT_SETTINGS_TYPE)
return boolean is

  OKAY  :  boolean := TRUE;

 begin

  case IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE is
   when STMT_TYPE'VAL (3) | STMT_TYPE'VAL (4) | STMT_TYPE'VAL (5) |
      STMT_TYPE'VAL (6) | STMT_TYPE'VAL (7) =>
     OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.THIRD_ATTRIBUTE is
   when ORGIN'VAL (5) | ORGIN'VAL (6) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.FOURTH_ATTRIBUTE is
   when USAGE'VAL (1) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.FIFTH_ATTRIBUTE is
   when DEVELOPMENT_STATUS'VAL (0) | DEVELOPMENT_STATUS'VAL (1) =>
    OKAY := FALSE;
   when others =>
```

```
      null;
    end case;

    return OKAY;

  end CHECKED_OKAY_B;

  --
  --
  function CHECKED_OKAY_C (IN_CURRENT_SETTINGS : CURRENT_SETTINGS_TYPE)
return boolean is

    OKAY   :  boolean := TRUE;

  begin

    case IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE is
      when STMT_TYPE'VAL (5) | STMT_TYPE'VAL (6) | STMT_TYPE'VAL (7) =>
        OKAY := FALSE;
      when others =>
        null;
    end case;

    case IN_CURRENT_SETTINGS.THIRD_ATTRIBUTE is
      when ORGIN'VAL (5) | ORGIN'VAL (6) =>
        OKAY := FALSE;
      when others =>
        null;
    end case;

    case IN_CURRENT_SETTINGS.FOURTH_ATTRIBUTE is
      when USAGE'VAL (1) =>
        OKAY := FALSE;
      when others =>
        null;
    end case;

    case IN_CURRENT_SETTINGS.FIFTH_ATTRIBUTE is
      when DEVELOPMENT_STATUS'VAL (0) | DEVELOPMENT_STATUS'VAL (1) |
          DEVELOPMENT_STATUS'VAL (2) | DEVELOPMENT_STATUS'VAL (3) |
          DEVELOPMENT_STATUS'VAL (4) | DEVELOPMENT_STATUS'VAL (5) |
          DEVELOPMENT_STATUS'VAL (6) =>
        OKAY := FALSE;
      when others =>
        null;
    end case;

    return OKAY;

  end CHECKED_OKAY_C;
```

```
--
--
function CHECKED_OKAY_D (IN_CURRENT_SETTINGS : CURRENT_SETTINGS_TYPE)
return boolean is

  OKAY  : boolean := TRUE;

begin

  case IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE is
    when STMT_TYPE'VAL (3) | STMT_TYPE'VAL (4) | STMT_TYPE'VAL (5) |
       STMT_TYPE'VAL (6) | STMT_TYPE'VAL (7) =>
      OKAY := FALSE;
    when others =>
      null;
  end case;

  case IN_CURRENT_SETTINGS.THIRD_ATTRIBUTE is
    when ORGIN'VAL (5) | ORGIN'VAL (6) =>
      OKAY := FALSE;
    when others =>
      null;
  end case;

  case IN_CURRENT_SETTINGS.FOURTH_ATTRIBUTE is
    when USAGE'VAL (1) =>
      OKAY := FALSE;
    when others =>
      null;
  end case;

  case IN_CURRENT_SETTINGS.FIFTH_ATTRIBUTE is
    when DEVELOPMENT_STATUS'VAL (0) | DEVELOPMENT_STATUS'VAL (1) |
       DEVELOPMENT_STATUS'VAL (2) | DEVELOPMENT_STATUS'VAL (3) |
       DEVELOPMENT_STATUS'VAL (4) | DEVELOPMENT_STATUS'VAL (5) |
       DEVELOPMENT_STATUS'VAL (6) =>
      OKAY := FALSE;
    when others =>
      null;
  end case;

  return OKAY;

end CHECKED_OKAY_D;


--
--
function CHECKED_OKAY_E (IN_CURRENT_SETTINGS : CURRENT_SETTINGS_TYPE)
return boolean is
```

```
  OKAY   :  boolean := TRUE;

 begin

  case IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE is
   when STMT_TYPE'VAL (5) | STMT_TYPE'VAL (6) | STMT_TYPE'VAL (7) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.THIRD_ATTRIBUTE is
   when ORGIN'VAL (5) | ORGIN'VAL (6) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.FOURTH_ATTRIBUTE is
   when USAGE'VAL (1) =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  case IN_CURRENT_SETTINGS.FIFTH_ATTRIBUTE is
   when DEVELOPMENT_STATUS'VAL (0) | DEVELOPMENT_STATUS'VAL (1) |
      DEVELOPMENT_STATUS'VAL (2) | DEVELOPMENT_STATUS'VAL (3) |
      DEVELOPMENT_STATUS'VAL (4) | DEVELOPMENT_STATUS'VAL (5) |
      DEVELOPMENT_STATUS'VAL (6)                    =>
    OKAY := FALSE;
   when others =>
    null;
  end case;

  return OKAY;

 end CHECKED_OKAY_E;



 --
 --
  function CHECKED_OKAY_F (IN_CURRENT_SETTINGS : CURRENT_SETTINGS_TYPE)
return boolean is

  OKAY   :  boolean := TRUE;

 begin
```

```
case IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE is
  when STMT_TYPE'VAL (0) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_1 then
      OKAY := FALSE;
    end if;
  when STMT_TYPE'VAL (1) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_3 then
      OKAY := FALSE;
    end if;
  when STMT_TYPE'VAL (2) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_4 then
      OKAY := FALSE;
    end if;
  when STMT_TYPE'VAL (3) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_6 then
      OKAY := FALSE;
    end if;
  when STMT_TYPE'VAL (4) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_7 then
      OKAY := FALSE;
    end if;
  when STMT_TYPE'VAL (5) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_8 then
      OKAY := FALSE;
    end if;
  when STMT_TYPE'VAL (6) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_9 then
      OKAY := FALSE;
    end if;
  when STMT_TYPE'VAL (7) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_10 then
      OKAY := FALSE;
    end if;
end case;

case IN_CURRENT_SETTINGS.SECOND_ATTRIBUTE is
  when HOW_PRODUCED'VAL (0) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL4.LINE_1 then
      OKAY := FALSE;
    end if;
  when HOW_PRODUCED'VAL (1) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL4.LINE_2 then
      OKAY := FALSE;
    end if;
  when HOW_PRODUCED'VAL (2) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL4.LINE_3 then
      OKAY := FALSE;
    end if;
  when HOW_PRODUCED'VAL (3) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL4.LINE_4 then
      OKAY := FALSE;
```

```
    end if;
   when HOW_PRODUCED'VAL (4) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL4.LINE_5 then
     OKAY := FALSE;
    end if;
   when HOW_PRODUCED'VAL (5) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL4.LINE_6 then
     OKAY := FALSE;
    end if;
  end case;


  case IN_CURRENT_SETTINGS.THIRD_ATTRIBUTE is
   when ORGIN'VAL (0) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_1 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (1) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_3 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (2) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_4 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (3) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_5 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (4) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_6 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (5) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_7 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (6) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_8 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (7) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_9 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (8) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_10 then
     OKAY := FALSE;
    end if;
   when ORGIN'VAL (9) =>
    if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_11 then
```

```
    OKAY := FALSE;
   end if;
  when ORGIN'VAL (10) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL5.LINE_12 then
    OKAY := FALSE;
   end if;
end case;

case IN_CURRENT_SETTINGS.FOURTH_ATTRIBUTE is
  when USAGE'VAL (0) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL6.LINE_1 then
    OKAY := FALSE;
   end if;
  when USAGE'VAL (1) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL6.LINE_2 then
    OKAY := FALSE;
   end if;
end case;

case IN_CURRENT_SETTINGS.FIFTH_ATTRIBUTE is
  when DEVELOPMENT_STATUS'VAL (0) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_1 then
    OKAY := FALSE;
   end if;
  when DEVELOPMENT_STATUS'VAL (1) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_2 then
    OKAY := FALSE;
   end if;
  when DEVELOPMENT_STATUS'VAL (2) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_3 then
    OKAY := FALSE;
   end if;
  when DEVELOPMENT_STATUS'VAL (3) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_4 then
    OKAY := FALSE;
   end if;
  when DEVELOPMENT_STATUS'VAL (4) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_5 then
    OKAY := FALSE;
   end if;
  when DEVELOPMENT_STATUS'VAL (5) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_6 then
    OKAY := FALSE;
   end if;
  when DEVELOPMENT_STATUS'VAL (6) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_7 then
    OKAY := FALSE;
   end if;
  when DEVELOPMENT_STATUS'VAL (7) =>
   if not GLOBAL.RECORD_FLAGS_F.PANEL9.LINE_8 then
    OKAY := FALSE;
```

```
    end if;
  end case;

  return OKAY;

end CHECKED_OKAY_F;


--
--
procedure PARSE_SPECIAL_LINE (LENGTH       : in    integer;
                              POINTER_1    : in out positive;
                              RETURN_VALUE :    out integer;
                              PARSE_STRING : in out string) is

  POINTER_2     : positive;
  NOT_FOUND     : boolean := FALSE;
  PARSE_STRING_2 : string (1..20) := (others => ' ');
  TEMP_INT      : integer;

begin

  while not NOT_FOUND and POINTER_1 <= (LENGTH - 2) loop
   if PARSE_STRING (POINTER_1 .. POINTER_1 + 3) = " => " then
    POINTER_1 := POINTER_1 + 4;
    POINTER_2 := POINTER_1 + 1;

    while not NOT_FOUND and POINTER_2 <= LENGTH loop
     if  PARSE_STRING (POINTER_2) = ',' or
       PARSE_STRING (POINTER_2) = ' '   then
      PARSE_STRING_2 (POINTER_1 - (POINTER_1 - 1) ..
                      (POINTER_2 - POINTER_1))
                := PARSE_STRING (POINTER_1 .. POINTER_2 - 1);
      TEMP_INT := integer'VALUE (PARSE_STRING_2);
      RETURN_VALUE := TEMP_INT;
      PARSE_STRING_2 := (others => ' ');
      NOT_FOUND := TRUE;
     else
      POINTER_2 := POINTER_2 + 1;
     end if;
    end loop;

   else
    POINTER_1 := POINTER_1 + 1;
   end if;

  end loop;

end PARSE_SPECIAL_LINE;

--
```

```
--
procedure PARSE_SPECIAL_COMMENT (IN_NUM    : in integer;
                                 IN_LENGTH : in integer;
                                 IN_STRING : in string) is

 TEMP_STRING      : string (1 .. 1024) := IN_STRING;
 EXEC_TEMP,
 DEC_TEMP,
 PRAGMA_TEMP      : natural := 0;
 OFFSET           : positive;
 OFFSET_1         : positive := 13;
 OFFSET_2         : positive := 26;
 OLD_SETTINGS     : CURRENT_SETTINGS_TYPE := CURRENT_SETTINGS;

begin

 if IN_NUM < 0 then
  OFFSET := OFFSET_1;
  CURRENT_SETTINGS.SECOND_ATTRIBUTE := HOW_PRODUCED'val (5);
 else
  OFFSET := OFFSET_2;
  CURRENT_SETTINGS.FIFTH_ATTRIBUTE := DEVELOPMENT_STATUS'val (0);
 end if;

 if IN_LENGTH > 0 then

  -- Now looking for the number associated with Executables
  PARSE_SPECIAL_LINE (IN_LENGTH,
                      OFFSET,
                      EXEC_TEMP,
                      TEMP_STRING);

  CURRENT_SETTINGS.FIRST_ATTRIBUTE := STMT_TYPE'val (0);
  if CHECK_REPORT_A_E then
   GLOBAL.DETERMINE_WHICH_ARRAY (IN_RECORD  => CURRENT_SETTINGS,
                                 ADD_NUMBER => EXEC_TEMP);
  end if;
  if CHECK_REPORT_F then
   if CHECKED_OKAY_F (CURRENT_SETTINGS) then
    COUNT_LINE (CURRENT_SETTINGS, COUNT_ARRAY_F, EXEC_TEMP);
   end if;
  end if;

  -- Now need to find the number associated for Declarations
  PARSE_SPECIAL_LINE (IN_LENGTH,
                      OFFSET,
                      DEC_TEMP,
                      TEMP_STRING);

  CURRENT_SETTINGS.FIRST_ATTRIBUTE := STMT_TYPE'val (1);
  if CHECK_REPORT_A_E then
```

```
      GLOBAL.DETERMINE_WHICH_ARRAY (IN_RECORD => CURRENT_SETTINGS,
                          ADD_NUMBER => DEC_TEMP);
     end if;
     if CHECK_REPORT_F then
      if CHECKED_OKAY_F (CURRENT_SETTINGS) then
       COUNT_LINE (CURRENT_SETTINGS, COUNT_ARRAY_F, DEC_TEMP);
      end if;
     end if;

     -- Now need to find the number associated for Compiler Directives
     PARSE_SPECIAL_LINE (IN_LENGTH,
                 OFFSET,
                 PRAGMA_TEMP,
                 TEMP_STRING);

     CURRENT_SETTINGS.FIRST_ATTRIBUTE := STMT_TYPE'val (2);
     if CHECK_REPORT_A_E then
      GLOBAL.DETERMINE_WHICH_ARRAY (IN_RECORD => CURRENT_SETTINGS,
                          ADD_NUMBER => PRAGMA_TEMP);
     end if;
     if CHECK_REPORT_F then
      if CHECKED_OKAY_F (CURRENT_SETTINGS) then
       COUNT_LINE (CURRENT_SETTINGS, COUNT_ARRAY_F, PRAGMA_TEMP);
      end if;
     end if;

     CURRENT_SETTINGS := OLD_SETTINGS;

    end if;

   end PARSE_SPECIAL_COMMENT;



   --
   --
   procedure DETERMINE_WHICH_ARRAY (IN_RECORD : in
CURRENT_SETTINGS_TYPE;
                          ADD_NUMBER : in    natural := 1 ) is

   begin

    --
    if RECORD_FLAGS.PANEL2.REPORT_A then
     if CHECKED_OKAY_A (IN_RECORD) then
      COUNT_LINE (IN_RECORD, COUNT_ARRAY_A, ADD_NUMBER);
     end if;
    end if;
    --
    if RECORD_FLAGS.PANEL2.REPORT_B then
     if CHECKED_OKAY_B (IN_RECORD) then
```

```
      COUNT_LINE (IN_RECORD, COUNT_ARRAY_B, ADD_NUMBER);
     end if;
    end if;
    --
    if RECORD_FLAGS.PANEL2.REPORT_C then
     if CHECKED_OKAY_C (IN_RECORD) then
      COUNT_LINE (IN_RECORD, COUNT_ARRAY_C, ADD_NUMBER);
     end if;
    end if;
    --
    if RECORD_FLAGS.PANEL2.REPORT_D then
     if CHECKED_OKAY_D (IN_RECORD) then
      COUNT_LINE (IN_RECORD, COUNT_ARRAY_D, ADD_NUMBER);
     end if;
    end if;
    --
    if RECORD_FLAGS.PANEL2.REPORT_E then
     if CHECKED_OKAY_E (IN_RECORD) then
      COUNT_LINE (IN_RECORD, COUNT_ARRAY_E, ADD_NUMBER);
     end if;
    end if;

  end DETERMINE_WHICH_ARRAY;



  --
  --
  procedure COUNT_LINE (IN_RECORD  : in    CURRENT_SETTINGS_TYPE;
                        ARRAY_TYPE : in out COUNT_ARRAY_TYPE;
                        ADD_NUMBER : in    natural := 1) is

   TEMP   :  natural := 0;

  begin

   TEMP := ARRAY_TYPE (IN_RECORD.FIRST_ATTRIBUTE,
          IN_RECORD.SECOND_ATTRIBUTE,
          IN_RECORD.THIRD_ATTRIBUTE,
          IN_RECORD.FOURTH_ATTRIBUTE,
             IN_RECORD.FIFTH_ATTRIBUTE);
   TEMP := TEMP + ADD_NUMBER;
--put (" +TEMP+");
--put (integer'image(temp));
   ARRAY_TYPE (IN_RECORD.FIRST_ATTRIBUTE,
         IN_RECORD.SECOND_ATTRIBUTE,
         IN_RECORD.THIRD_ATTRIBUTE,
         IN_RECORD.FOURTH_ATTRIBUTE,
         IN_RECORD.FIFTH_ATTRIBUTE) := TEMP;
  end COUNT_LINE;
```

```
--
--
 procedure ADD_TO_ARRAY_A_E (IN_CURRENT_SETTINGS : in out
CURRENT_SETTINGS_TYPE) is

  ADDED_TO_ARRAY    :  boolean := false;

 begin

     for I in PRIORITY_ARRAY_A_E'range loop

      for J in STMT_TYPE loop

        -- Found the statement type with the highest priority
        -- Conditions:
        -- Start flag is true, stop flag is true, ADDED_TO_ARRAY is false.
    -- Actions:
    -- Set ADDED_TO_ARRAY to true; set the FIRST_ATTRIBUTE to the
    -- current STATEMENT_TYPE (J);  set both the
    -- start and stop flag of the current statement type to false.
    if PRIORITY_ARRAY_A_E (I) = J  and
       FLAGS_ARRAY (J, 1)        and
       FLAGS_ARRAY (J, 2)        and
       not (ADDED_TO_ARRAY)         then

         IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE := J;
         DETERMINE_WHICH_ARRAY (IN_RECORD => IN_CURRENT_SETTINGS);
      ADDED_TO_ARRAY := true;

        -- Found the statement type with the highest priority
    -- that extends over two or more lines.
    -- Conditions:
    -- Start flag is true, stop flag is false, ADDED_TO_ARRAY is false.
    -- Actions:
    -- Set ADDED_TO_ARRAY to true; set the FIRST_ATTRIBUTE to the
    -- current STATEMENT_TYPE (J);
    elsif PRIORITY_ARRAY_A_E (I) = J      and
                FLAGS_ARRAY (J,1)     and
           not FLAGS_ARRAY(J,2)  and
           not ADDED_TO_ARRAY        then
      IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE := J;
      ADDED_TO_ARRAY := TRUE;
         DETERMINE_WHICH_ARRAY (IN_RECORD => IN_CURRENT_SETTINGS);
    end if;

      end loop;

     end loop;
```

```
 end ADD_TO_ARRAY_A_E;

 --
 --
 procedure ADD_TO_ARRAY_F (IN_CURRENT_SETTINGS : in out
CURRENT_SETTINGS_TYPE) is

  ADDED_TO_ARRAY    : boolean := false;

 begin

    for I in PRIORITY_ARRAY_F'range loop

     for J in STMT_TYPE loop

       -- Found the statement type with the highest priority
       -- Conditions:
   -- Start flag is true, stop flag is true, ADDED_TO_ARRAY is false.
   -- Actions:
   -- Set ADDED_TO_ARRAY to true; set the FIRST_ATTRIBUTE to the
   -- current STATEMENT_TYPE (J); call the procedure COUNT_LINE
   -- which will add this line to the total; set both the
   -- start and stop flag of the current statement type to false.
   if PRIORITY_ARRAY_F (I) = J  and
         FLAGS_ARRAY (J, 1)      and
     FLAGS_ARRAY (J, 2)      and
     not (ADDED_TO_ARRAY)        then

       IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE := J;
       if CHECKED_OKAY_F (IN_CURRENT_SETTINGS) then
        GLOBAL.COUNT_LINE (IN_CURRENT_SETTINGS, COUNT_ARRAY_F);
       end if;
    ADDED_TO_ARRAY := true;

   -- Found the statement type with the highest priority
   -- that extends over two or more lines.
   -- Conditions:
   -- Start flag is true, stop flag is false, ADDED_TO_ARRAY is false.
   -- Actions:
   -- Set ADDED_TO_ARRAY to true; set the FIRST_ATTRIBUTE to the
   -- current STATEMENT_TYPE (J); and call the procedure COUNT_LINE
      -- which will add this line to the total.
   elsif PRIORITY_ARRAY_F (I) = J       and
              FLAGS_ARRAY (J,1)    and
         not FLAGS_ARRAY(J,2)  and
         not ADDED_TO_ARRAY        then
    IN_CURRENT_SETTINGS.FIRST_ATTRIBUTE := J;
    ADDED_TO_ARRAY := TRUE;
       if CHECKED_OKAY_F (IN_CURRENT_SETTINGS) then
        GLOBAL.COUNT_LINE (IN_CURRENT_SETTINGS, COUNT_ARRAY_F);
       end if;
```

```
    end if;

      end loop;

    end loop;

end ADD_TO_ARRAY_F;

--
--
procedure ADD_TO_ARRAY is

 CURRENT_SETTINGS_A_E,
 CURRENT_SETTINGS_F   : CURRENT_SETTINGS_TYPE := CURRENT_SETTINGS;

begin

 if not SPECIAL_COMMENT then

  if  FLAGS_ARRAY (STMT_TYPE'val (0), 1)     and
      not FLAGS_ARRAY (STMT_TYPE'val (0), 2)     then
   EXECLEVEL := EXECLEVEL + 1;
  elsif not FLAGS_ARRAY (STMT_TYPE'val (0), 1) and
      FLAGS_ARRAY (STMT_TYPE'val (0), 2)        then
   FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
   EXECLEVEL := EXECLEVEL - 1;
   if EXECLEVEL < 0 then
    put_line("Warning: execution parsing is confused");
   end if;
  elsif EXECLEVEL > 0 then
   FLAGS_ARRAY (STMT_TYPE'val (0), 1) := TRUE;
  else
   null;
  end if;

  if  FLAGS_ARRAY (STMT_TYPE'val (1), 1)     and
      not FLAGS_ARRAY (STMT_TYPE'val (1), 2)     then
   DECLEVEL := DECLEVEL + 1;
  elsif not FLAGS_ARRAY (STMT_TYPE'val (1), 1) and
      FLAGS_ARRAY (STMT_TYPE'val (1), 2)        then
   FLAGS_ARRAY (STMT_TYPE'val (1), 1) := TRUE;
   DECLEVEL := DECLEVEL - 1;
   if DECLEVEL < 0 then
    put_line("Warning: declaration parsing is confused");
   end if;
  elsif DECLEVEL > 0 then
   FLAGS_ARRAY (STMT_TYPE'val (1), 1) := TRUE;
  else
   null;
  end if;
```

```
    if  FLAGS_ARRAY (STMT_TYPE'val (2), 1)     and
        not FLAGS_ARRAY (STMT_TYPE'val (2), 2)     then
      PRAGMALEVEL := PRAGMALEVEL + 1;
    elsif not FLAGS_ARRAY (STMT_TYPE'val (2), 1) and
         FLAGS_ARRAY (STMT_TYPE'val (2), 2)       then
      FLAGS_ARRAY (STMT_TYPE'val (2), 1) := TRUE;
      PRAGMALEVEL := PRAGMALEVEL - 1;
    elsif PRAGMALEVEL > 0 then
      FLAGS_ARRAY (STMT_TYPE'val (2), 1) := TRUE;
    else
      null;
    end if;
-- Debugging statements to help figure out the three
--  variables used to track multiline flags
--put ("[");
--put (integer'image(execlevel));
--put ("]");
--put ("[");
--put (integer'image(declevel));
--put ("]");
--put ("[");
--put (integer'image(pragmalevel));
--put ("]");

    --
    -- Checking for full line code
    if  FLAGS_ARRAY (STMT_TYPE'val (3), 1)   then
      CURRENT_SETTINGS.FIRST_ATTRIBUTE := STMT_TYPE'val (3);
      DETERMINE_WHICH_ARRAY (IN_RECORD => CURRENT_SETTINGS);

      if CHECK_REPORT_F then
        if CHECKED_OKAY_F (CURRENT_SETTINGS) then
          GLOBAL.COUNT_LINE (CURRENT_SETTINGS, COUNT_ARRAY_F);
        end if;
      end if;

    elsif FLAGS_ARRAY (STMT_TYPE'val (5), 1)  then
      CURRENT_SETTINGS.FIRST_ATTRIBUTE := STMT_TYPE'val (5);
      DETERMINE_WHICH_ARRAY (IN_RECORD => CURRENT_SETTINGS);

      if CHECK_REPORT_F then
        if CHECKED_OKAY_F (CURRENT_SETTINGS) then
          GLOBAL.COUNT_LINE (CURRENT_SETTINGS, COUNT_ARRAY_F);
        end if;
      end if;

    elsif FLAGS_ARRAY (STMT_TYPE'val (6), 1)  then
      CURRENT_SETTINGS.FIRST_ATTRIBUTE := STMT_TYPE'val (6);
      DETERMINE_WHICH_ARRAY (IN_RECORD => CURRENT_SETTINGS);
```

```
    if CHECK_REPORT_F then
     if CHECKED_OKAY_F (CURRENT_SETTINGS) then
      GLOBAL.COUNT_LINE (CURRENT_SETTINGS, COUNT_ARRAY_F);
     end if;
    end if;

   elsif FLAGS_ARRAY (STMT_TYPE'val (7), 1)  then
    CURRENT_SETTINGS.FIRST_ATTRIBUTE := STMT_TYPE'val (7);
    DETERMINE_WHICH_ARRAY (IN_RECORD => CURRENT_SETTINGS);

    if CHECK_REPORT_F then
     if CHECKED_OKAY_F (CURRENT_SETTINGS) then
      GLOBAL.COUNT_LINE (CURRENT_SETTINGS, COUNT_ARRAY_F);
     end if;
    end if;

   else

    if  CHECK_REPORT_A_E then
     ADD_TO_ARRAY_A_E (CURRENT_SETTINGS_A_E);
    end if;

    if CHECK_REPORT_F then
     ADD_TO_ARRAY_F (CURRENT_SETTINGS_F);
    end if;

   end if;

  else

   SPECIAL_COMMENT := FALSE;

  end if;

  -- for insurance, clear appropriate flags before processing the next line
  for i in STMT_TYPE'first .. STMT_TYPE'last loop
   FLAGS_ARRAY(I,1) := FALSE;
   FLAGS_ARRAY(I,2) := FALSE;
  end loop;

end ADD_TO_ARRAY;

--
--
procedure INIT_RECORD_FLAGS is
begin

 RECORD_FLAGS.PANEL6.DEL_OPTION (1)(my_value'range) :=  (others => ' ');
 RECORD_FLAGS.PANEL6.DEL_OPTION (1)(1 .. 19) := "Delivered as source";

end INIT_RECORD_FLAGS;
```

```
--
--
procedure INIT_RECORD_FLAGS_B is
begin

  RECORD_FLAGS_B.PANEL4.LINE_6 := TRUE;
  RECORD_FLAGS_B.PANEL9.LINE_3 := TRUE;
  RECORD_FLAGS_B.PANEL9.LINE_4 := TRUE;
  RECORD_FLAGS_B.PANEL9.LINE_5 := TRUE;
  RECORD_FLAGS_B.PANEL9.LINE_6 := TRUE;
  RECORD_FLAGS_B.PANEL9.LINE_7 := TRUE;

end INIT_RECORD_FLAGS_B;

--
--
procedure INIT_RECORD_FLAGS_C is
begin

  RECORD_FLAGS_C.PANEL3.LINE_6 := TRUE;
  RECORD_FLAGS_C.PANEL3.LINE_7 := TRUE;
  RECORD_FLAGS_C.PANEL4.LINE_6 := TRUE;

end INIT_RECORD_FLAGS_C;


--
--
procedure INIT_RECORD_FLAGS_D is
begin

  RECORD_FLAGS_D.PANEL4.LINE_6 := TRUE;

end INIT_RECORD_FLAGS_D;


--
--
procedure INIT_RECORD_FLAGS_E is
begin

  RECORD_FLAGS_E.PANEL3.LINE_6 := TRUE;
  RECORD_FLAGS_E.PANEL3.LINE_7 := TRUE;
  RECORD_FLAGS_E.PANEL4.LINE_6 := TRUE;

end INIT_RECORD_FLAGS_E;


--
--
```

```
  procedure INIT_RECORD_FLAGS_F is
  begin

    RECORD_FLAGS_F := RECORD_FLAGS;

  end INIT_RECORD_FLAGS_F;


begin

  INIT_RECORD_FLAGS;
  INIT_RECORD_FLAGS_B;
  INIT_RECORD_FLAGS_C;
  INIT_RECORD_FLAGS_D;
  INIT_RECORD_FLAGS_E;
  INIT_RECORD_FLAGS_F;

end Global;
```

COUNT_TOOL_PKG

```
--*_Programmed
with GLOBAL,
   TAE,
   PARSER,
   ADA_LEX_IO,
   ADA_LEX,
   REPORT_PACKAGE,
   TEXT_IO;
use  TAE,
   TEXT_IO;
--
--
package TOOL_PACKAGE is


  procedure SET_PRECEDENCE_F;

  procedure SET_PRECEDENCE_A_E;

  procedure START_COUNT;


end TOOL_PACKAGE;


--*_Programmed
--
--
package body TOOL_PACKAGE is

  INITIAL_TYPE      : GLOBAL.STMT_TYPE;
  INITIAL_PRIORITY   : TAE.TAEINT;

  package TAE_INTEGER_IN_OUT is new integer_io (TAE.TAEINT);
  use TAE_INTEGER_IN_OUT;

  package INTEGER_IN_OUT is new integer_io (integer);
  use INTEGER_IN_OUT;

  package ENUMERATION_IN_OUT is new ENUMERATION_IO (GLOBAL.STMT_TYPE);
  use ENUMERATION_IN_OUT;

  --
  --
  procedure DETERMINE_PRIORITY (A  : in    GLOBAL.STMT_TYPE;
                    B  : in    TAE.TAEINT;
                    C  : in out GLOBAL.STMT_TYPE;
```

```
                    D  : in out TAE.TAEINT) is

begin

 if (D > B and B /= 0) or D = 0 then
  C := A;
  D := B;
 end if;

end DETERMINE_PRIORITY;

--
--
procedure SET_PRECEDENCE_F is

 TEMP_TYPE   : GLOBAL.STMT_TYPE;

begin

 FOR I in GLOBAL.PRIORITY_ARRAY_F'range loop

  INITIAL_TYPE := GLOBAL.STMT_TYPE'val(0);
  INITIAL_PRIORITY := GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_1_INT;

  DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(0),
              GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_1_INT,
              INITIAL_TYPE,
              INITIAL_PRIORITY);
  DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(1),
              GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_3_INT,
              INITIAL_TYPE,
              INITIAL_PRIORITY);
  DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(2),
              GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_4_INT,
              INITIAL_TYPE,
              INITIAL_PRIORITY);
  DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(3),
              GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_6_INT,
              INITIAL_TYPE,
              INITIAL_PRIORITY);
  DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(4),
              GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_7_INT,
              INITIAL_TYPE,
              INITIAL_PRIORITY);
  DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(5),
              GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_8_INT,
              INITIAL_TYPE,
              INITIAL_PRIORITY);
  DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(6),
              GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_9_INT,
              INITIAL_TYPE,
```

```
                INITIAL_PRIORITY);
    DETERMINE_PRIORITY (GLOBAL.STMT_TYPE'val(7),
                GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_10_INT,
                INITIAL_TYPE,
                INITIAL_PRIORITY);

    case INITIAL_TYPE is
     when GLOBAL.STMT_TYPE'val(0) =>
      new_line (2);
--      put (GLOBAL.STMT_TYPE'val(0));
--      put (" is priority: ");
--      put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_1_INT);
--      new_line;
      GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_1_INT := 0;
     when GLOBAL.STMT_TYPE'val(1) =>
--      put (GLOBAL.STMT_TYPE'val(1));
--      put (" is priority: ");
--      put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_3_INT);
--      new_line;
      GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_3_INT := 0;
     when GLOBAL.STMT_TYPE'val(2) =>
--      put (GLOBAL.STMT_TYPE'val(2));
--      put (" is priority: ");
--      put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_4_INT);
--      new_line;
      GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_4_INT := 0;
     when GLOBAL.STMT_TYPE'val(3) =>
--      put (GLOBAL.STMT_TYPE'val(3));
--      put (" is priority: ");
--      put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_6_INT);
--      new_line;
      GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_6_INT := 0;
     when GLOBAL.STMT_TYPE'val(4) =>
--      put (GLOBAL.STMT_TYPE'val(4));
--      put (" is priority: ");
--      put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_7_INT);
--      new_line;
      GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_7_INT := 0;
     when GLOBAL.STMT_TYPE'val(5) =>
--      put (GLOBAL.STMT_TYPE'val(5));
--      put (" is priority: ");
--      put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_8_INT);
--      new_line;
      GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_8_INT := 0;
     when GLOBAL.STMT_TYPE'val(6) =>
--      put (GLOBAL.STMT_TYPE'val(6));
--      put (" is priority: ");
--      put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_9_INT);
--      new_line;
      GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_9_INT := 0;
     when GLOBAL.STMT_TYPE'val(7) =>
```

```
--        put (GLOBAL.STMT_TYPE'val(7));
--        put (" is priority: ");
--        put (GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_10_INT);
--        new_line (2);
        GLOBAL.RECORD_FLAGS_F.PANEL3.LINE_10_INT := 0;
      end case;

    GLOBAL.PRIORITY_ARRAY_F (I) := INITIAL_TYPE;

  end loop;

end SET_PRECEDENCE_F;


--
--
procedure SET_PRECEDENCE_A_E is

begin

    GLOBAL.PRIORITY_ARRAY_A_E (1) := GLOBAL.STMT_TYPE'val(0);
    GLOBAL.PRIORITY_ARRAY_A_E (2) := GLOBAL.STMT_TYPE'val(1);
    GLOBAL.PRIORITY_ARRAY_A_E (3) := GLOBAL.STMT_TYPE'val(2);
    GLOBAL.PRIORITY_ARRAY_A_E (4) := GLOBAL.STMT_TYPE'val(3);
    GLOBAL.PRIORITY_ARRAY_A_E (5) := GLOBAL.STMT_TYPE'val(4);
    GLOBAL.PRIORITY_ARRAY_A_E (6) := GLOBAL.STMT_TYPE'val(5);
    GLOBAL.PRIORITY_ARRAY_A_E (7) := GLOBAL.STMT_TYPE'val(6);
    GLOBAL.PRIORITY_ARRAY_A_E (8) := GLOBAL.STMT_TYPE'val(7);

end SET_PRECEDENCE_A_E;



--
--
procedure GET_FILE_NAME (OUT_FILE_NAME  :   out string;
                         OUT_NAME_LENGTH :   out integer;
                         F               : in out file_type) is

  TEMP_NUMBER   : integer := 0;
  TEMP_NAME     : string (1 .. 80) := (others => ' ');
  NOT_BLANK     : boolean := TRUE;

begin

  if not end_of_file (F) then
    get_line (F, TEMP_NAME, TEMP_NUMBER);

    while NOT_BLANK loop
      if TEMP_NAME (TEMP_NUMBER) = ' ' then
        TEMP_NUMBER := TEMP_NUMBER - 1;
```

```
      else
        NOT_BLANK := FALSE;
      end if;
     end loop;

    end if;

    OUT_FILE_NAME := TEMP_NAME;
    OUT_NAME_LENGTH := TEMP_NUMBER;

--put ("the name of the file to opened is: ");
--put_line (temp_name);
--put ("the file name is this long:  ");
--put (temp_number, width => 3);
--new_line;

 end GET_FILE_NAME;


 --
 --
 procedure START_PARSE is

  IS_FILELIST      : boolean;
  FILELIST         : GLOBAL.MY_VALUE := GLOBAL.RECORD_FLAGS.
                                PANEL2.IN_FILE_NAME;
  LENGTH           : integer;
  FILE_NAME        : string(1..80);
  LAST             : integer := 0;
  FILE_LIST_NAME   : string (1..1024) := (others => ' ');
  F                : file_type;

 begin

  LENGTH := GLOBAL.FIND_LENGTH (FILELIST);

  FILE_LIST_NAME(1..LENGTH) := FILELIST (1) (1..LENGTH);

  open (F, in_file, FILE_LIST_NAME (1..LENGTH));

  while not END_OF_FILE (F) loop

   GET_FILE_NAME (FILE_NAME, LAST, F);

   ADA_LEX_IO.OPEN_INPUT (FILE_NAME (1..LAST));
   ADA_LEX_IO.CREATE_OUTPUT;

   new_line;
   put_line ("  Starting parse  ");
   ADA_LEX.linenum;
   PARSER.yyparse;
```

```
    new_line;
    put_line (" Finished parse ");

    ADA_LEX_IO.CLOSE_INPUT;
    ADA_LEX_IO.CLOSE_OUTPUT;

   end loop;

  end START_PARSE;


  --
  --
  procedure START_COUNT is

  begin

   SET_PRECEDENCE_A_E;

   if GLOBAL.RECORD_FLAGS.PANEL2.REPORT_F then
    SET_PRECEDENCE_F;
   end if;

   GLOBAL.OPEN_OUT_FILE;

   START_PARSE;

   REPORT_PACKAGE.DETERMINE_WHICH_REPORT;

  end START_COUNT;

end TOOL_PACKAGE;
```

REPORT_PKG_S.A

```
--*_Programmed

with GLOBAL,
   TAE,
   TEXT_IO,
   GENERIC_COUNTS;
use TAE,
   GLOBAL,
   TEXT_IO;

--
--
package REPORT_PACKAGE is

 type TYPE_NUMBER_TYPE is range 1 .. 5;
 T2_NUMBER  : TYPE_NUMBER_TYPE;

 type NAME_REPORT_TYPE is (REPORT_A, REPORT_B, REPORT_C,
                REPORT_D, REPORT_E, REPORT_F);
 REPORT_NAME : NAME_REPORT_TYPE;

 procedure DETERMINE_WHICH_REPORT;

 function RETRIEVE_2D_1 (TYPE_1 : STMT_TYPE;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : ORGIN;
                TYPE_4 : USAGE;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

 function RETRIEVE_2D_2 (TYPE_1 : STMT_TYPE;
                TYPE_2 : USAGE;
                TYPE_3 : HOW_PRODUCED;
                TYPE_4 : ORGIN;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

 function RETRIEVE_2D_3 (TYPE_1 : ORGIN;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : USAGE;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

 function RETRIEVE_2D_4 (TYPE_1 : ORGIN;
                TYPE_2 : STMT_TYPE;
                TYPE_3 : HOW_PRODUCED;
```

```
                TYPE_4 : USAGE;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_2D_5 (TYPE_1 : ORGIN;
                TYPE_2 : USAGE;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : HOW_PRODUCED;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_2D_6 (TYPE_1 : ORGIN;
                TYPE_2 : DEVELOPMENT_STATUS;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : HOW_PRODUCED;
                TYPE_5 : USAGE;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_2D_7 (TYPE_1 : USAGE;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : ORGIN;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_2D_8 (TYPE_1 : DEVELOPMENT_STATUS;
                TYPE_2 : STMT_TYPE;
                TYPE_3 : HOW_PRODUCED;
                TYPE_4 : ORGIN;
                TYPE_5 : USAGE;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_2D_9 (TYPE_1 : DEVELOPMENT_STATUS;
                TYPE_2 : USAGE;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : HOW_PRODUCED;
                TYPE_5 : ORGIN;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_2D_10 (TYPE_1 : DEVELOPMENT_STATUS;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : ORGIN;
                TYPE_5 : USAGE;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_3D_1 (TYPE_1 : USAGE;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : ORGIN;
                TYPE_4 : STMT_TYPE;
                TYPE_5 : DEVELOPMENT_STATUS;
```

```
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_3D_2 (TYPE_1 : USAGE;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : DEVELOPMENT_STATUS;
                TYPE_4 : ORGIN;
                TYPE_5 : STMT_TYPE;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_3D_3 (TYPE_1 : DEVELOPMENT_STATUS;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : ORGIN;
                TYPE_4 : STMT_TYPE;
                TYPE_5 : USAGE;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function RETRIEVE_3D_4 (TYPE_1 : DEVELOPMENT_STATUS;
                TYPE_2 : USAGE;
                TYPE_3 : ORGIN;
                TYPE_4 : HOW_PRODUCED;
                TYPE_5 : STMT_TYPE;
                TYPE_6 : NAME_REPORT_TYPE) return natural;

function CHECK_2D_1 return TYPE_NUMBER_TYPE;

function CHECK_2D_2 return TYPE_NUMBER_TYPE;

function CHECK_2D_3 return TYPE_NUMBER_TYPE;

function CHECK_2D_4 return TYPE_NUMBER_TYPE;

function CHECK_3D_1 return TYPE_NUMBER_TYPE;

function CHECK_3D_2 return TYPE_NUMBER_TYPE;

function CHECK_3D_3 return TYPE_NUMBER_TYPE;

function CHECK_3D_4 return TYPE_NUMBER_TYPE;

procedure HEADING_STMT_TYPE (ROW_POSITION : positive);

procedure HEADING_ORGIN (ROW_POSITION : positive);

procedure HEADING_USAGE (ROW_POSITION : positive);

procedure HEADING_DEV_STATUS (ROW_POSITION : positive);

package INTEGER_IN_OUT is new integer_io (integer);
use INTEGER_IN_OUT;

package ENUMERATION_IN_OUT is new ENUMERATION_IO (STMT_TYPE);
```

```
use ENUMERATION_IN_OUT;

package RETRIEVE_1_2D is new GENERIC_COUNTS (FIRST_TYPE      => STMT_TYPE,
                        SECOND_TYPE     => HOW_PRODUCED,
                        THIRD_TYPE      => ORGIN,
                        FOURTH_TYPE     => USAGE,
                        FIFTH_TYPE      => DEVELOPMENT_STATUS,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_1,
                        CHECK_TYPE_2    => CHECK_2D_1,
                        CHECK_TYPE_3    => CHECK_3D_1,
                        PRINT_ROW_HEADING => HEADING_STMT_TYPE);

package RETRIEVE_2_2D is new GENERIC_COUNTS (FIRST_TYPE      => STMT_TYPE,
                        SECOND_TYPE     => USAGE,
                        THIRD_TYPE      => HOW_PRODUCED,
                        FOURTH_TYPE     => ORGIN,
                        FIFTH_TYPE      => DEVELOPMENT_STATUS,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_2,
                        CHECK_TYPE_2    => CHECK_2D_3,
                        CHECK_TYPE_3    => CHECK_3D_1,
                        PRINT_ROW_HEADING => HEADING_STMT_TYPE);

package RETRIEVE_3_2D is new GENERIC_COUNTS (FIRST_TYPE      => ORGIN,
                        SECOND_TYPE     => HOW_PRODUCED,
                        THIRD_TYPE      => STMT_TYPE,
                        FOURTH_TYPE     => USAGE,
                        FIFTH_TYPE      => DEVELOPMENT_STATUS,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_3,
                        CHECK_TYPE_2    => CHECK_2D_1,
                        CHECK_TYPE_3    => CHECK_3D_2,
                        PRINT_ROW_HEADING => HEADING_ORGIN);

package RETRIEVE_4_2D is new GENERIC_COUNTS (FIRST_TYPE      => ORGIN,
                        SECOND_TYPE     => STMT_TYPE,
                        THIRD_TYPE      => HOW_PRODUCED,
                        FOURTH_TYPE     => USAGE,
                        FIFTH_TYPE      => DEVELOPMENT_STATUS,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_4,
                        CHECK_TYPE_2    => CHECK_2D_2,
                        CHECK_TYPE_3    => CHECK_3D_3,
                        PRINT_ROW_HEADING => HEADING_ORGIN);

package RETRIEVE_5_2D is new GENERIC_COUNTS (FIRST_TYPE      => ORGIN,
```

```
                        SECOND_TYPE     => USAGE,
                        THIRD_TYPE      => STMT_TYPE,
                        FOURTH_TYPE     => HOW_PRODUCED,
                        FIFTH_TYPE      => DEVELOPMENT_STATUS,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_5,
                        CHECK_TYPE_2    => CHECK_2D_3,
                        CHECK_TYPE_3    => CHECK_3D_2,
                        PRINT_ROW_HEADING => HEADING_ORGIN);

 package RETRIEVE_6_2D is new GENERIC_COUNTS (FIRST_TYPE      => ORGIN,
                        SECOND_TYPE     => DEVELOPMENT_STATUS,
                        THIRD_TYPE      => STMT_TYPE,
                        FOURTH_TYPE     => HOW_PRODUCED,
                        FIFTH_TYPE      => USAGE,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_6,
                        CHECK_TYPE_2    => CHECK_2D_4,
                        CHECK_TYPE_3    => CHECK_3D_2,
                        PRINT_ROW_HEADING => HEADING_ORGIN);

 package RETRIEVE_7_2D is new GENERIC_COUNTS (FIRST_TYPE      => USAGE,
                        SECOND_TYPE     => HOW_PRODUCED,
                        THIRD_TYPE      => STMT_TYPE,
                        FOURTH_TYPE     => ORGIN,
                        FIFTH_TYPE      => DEVELOPMENT_STATUS,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_7,
                        CHECK_TYPE_2    => CHECK_2D_1,
                        CHECK_TYPE_3    => CHECK_3D_2,
                        PRINT_ROW_HEADING => HEADING_USAGE);

 package RETRIEVE_8_2D is new GENERIC_COUNTS (FIRST_TYPE      =>
DEVELOPMENT_STATUS,
                        SECOND_TYPE     => STMT_TYPE,
                        THIRD_TYPE      => HOW_PRODUCED,
                        FOURTH_TYPE     => ORGIN,
                        FIFTH_TYPE      => USAGE,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_8,
                        CHECK_TYPE_2    => CHECK_2D_2,
                        CHECK_TYPE_3    => CHECK_3D_3,
                        PRINT_ROW_HEADING => HEADING_DEV_STATUS);

 package RETRIEVE_9_2D is new GENERIC_COUNTS (FIRST_TYPE      =>
DEVELOPMENT_STATUS,
                        SECOND_TYPE     => USAGE,
```

```
                        THIRD_TYPE      => STMT_TYPE,
                        FOURTH_TYPE     => HOW_PRODUCED,
                        FIFTH_TYPE      => ORGIN,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_9,
                        CHECK_TYPE_2    => CHECK_2D_3,
                        CHECK_TYPE_3    => CHECK_3D_2,
                        PRINT_ROW_HEADING => HEADING_DEV_STATUS);

 package RETRIEVE_10_2D is new GENERIC_COUNTS (FIRST_TYPE      =>
DEVELOPMENT_STATUS,
                        SECOND_TYPE     => HOW_PRODUCED,
                        THIRD_TYPE      => STMT_TYPE,
                        FOURTH_TYPE     => ORGIN,
                        FIFTH_TYPE      => USAGE,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_2D_10,
                        CHECK_TYPE_2    => CHECK_2D_1,
                        CHECK_TYPE_3    => CHECK_3D_2,
                        PRINT_ROW_HEADING => HEADING_DEV_STATUS);

 package RETRIEVE_1_3D is new GENERIC_COUNTS (FIRST_TYPE      => USAGE,
                        SECOND_TYPE     => HOW_PRODUCED,
                        THIRD_TYPE      => ORGIN,
                        FOURTH_TYPE     => STMT_TYPE,
                        FIFTH_TYPE      => DEVELOPMENT_STATUS,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_3D_1,
                        CHECK_TYPE_2    => CHECK_2D_1,
                        CHECK_TYPE_3    => CHECK_3D_1,
                        PRINT_ROW_HEADING => HEADING_USAGE);

 package RETRIEVE_2_3D is new GENERIC_COUNTS (FIRST_TYPE      => USAGE,
                        SECOND_TYPE     => HOW_PRODUCED,
                        THIRD_TYPE      => DEVELOPMENT_STATUS,
                        FOURTH_TYPE     => ORGIN,
                        FIFTH_TYPE      => STMT_TYPE,
                        REPORT_TYPE     => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE   => TYPE_NUMBER_TYPE,
                        RETRIEVE        => RETRIEVE_3D_2,
                        CHECK_TYPE_2    => CHECK_2D_1,
                        CHECK_TYPE_3    => CHECK_3D_4,
                        PRINT_ROW_HEADING => HEADING_USAGE);

 package RETRIEVE_3_3D is new GENERIC_COUNTS (FIRST_TYPE      =>
DEVELOPMENT_STATUS,
                        SECOND_TYPE     => HOW_PRODUCED,
                        THIRD_TYPE      => ORGIN,
```

```
                        FOURTH_TYPE      => STMT_TYPE,
                        FIFTH_TYPE       => USAGE,
                        REPORT_TYPE      => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE    => TYPE_NUMBER_TYPE,
                        RETRIEVE         => RETRIEVE_3D_3,
                        CHECK_TYPE_2     => CHECK_2D_1,
                        CHECK_TYPE_3     => CHECK_3D_1,
                        PRINT_ROW_HEADING => HEADING_DEV_STATUS);

 package RETRIEVE_4_3D is new GENERIC_COUNTS (FIRST_TYPE       =>
DEVELOPMENT_STATUS,
                        SECOND_TYPE      => USAGE,
                        THIRD_TYPE       => ORGIN,
                        FOURTH_TYPE      => HOW_PRODUCED,
                        FIFTH_TYPE       => STMT_TYPE,
                        REPORT_TYPE      => NAME_REPORT_TYPE,
                        T_NUMBER_TYPE    => TYPE_NUMBER_TYPE,
                        RETRIEVE         => RETRIEVE_3D_4,
                        CHECK_TYPE_2     => CHECK_2D_3,
                        CHECK_TYPE_3     => CHECK_3D_1,
                        PRINT_ROW_HEADING => HEADING_DEV_STATUS);

end REPORT_PACKAGE;
```

REPORT_PKG_B.A

```
--*_Programmed

--
--
package body REPORT_PACKAGE is


--
--
function CNT_EST (D            : in DEVELOPMENT_STATUS;
                  IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT  : integer := 0;

begin

 for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
  for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
   for O in ORGIN'FIRST .. ORGIN'LAST loop
    for U in USAGE'FIRST .. USAGE'LAST loop

     TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);

    end loop;
   end loop;
  end loop;
 end loop;

 return TEMP_COUNT;

end CNT_EST;


 --
 --
 function RETRIEVE_2D_1 (TYPE_1  :  STMT_TYPE;
                         TYPE_2  :  HOW_PRODUCED;
                         TYPE_3  :  ORGIN;
                         TYPE_4  :  USAGE;
                         TYPE_5  :  DEVELOPMENT_STATUS;
                         TYPE_6  :  NAME_REPORT_TYPE) return natural is

  TEMP    :  natural := 0;

 begin

  case TYPE_6 is
```

```
   when REPORT_A =>
    TEMP := COUNT_ARRAY_A (TYPE_1, TYPE_2, TYPE_3, TYPE_4, TYPE_5);
   when REPORT_B =>
    TEMP := COUNT_ARRAY_B (TYPE_1, TYPE_2, TYPE_3, TYPE_4, TYPE_5);
   when REPORT_C =>
    TEMP := COUNT_ARRAY_C (TYPE_1, TYPE_2, TYPE_3, TYPE_4, TYPE_5);
   when REPORT_D =>
    TEMP := COUNT_ARRAY_D (TYPE_1, TYPE_2, TYPE_3, TYPE_4, TYPE_5);
   when REPORT_E =>
    TEMP := COUNT_ARRAY_E (TYPE_1, TYPE_2, TYPE_3, TYPE_4, TYPE_5);
   when REPORT_F =>
    TEMP := COUNT_ARRAY_F (TYPE_1, TYPE_2, TYPE_3, TYPE_4, TYPE_5);
  end case;

  return TEMP;

end RETRIEVE_2D_1;

--
--
function RETRIEVE_2D_2 (TYPE_1 : STMT_TYPE;
                TYPE_2 : USAGE;
                TYPE_3 : HOW_PRODUCED;
                TYPE_4 : ORGIN;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

  TEMP    : natural := 0;

begin

  case TYPE_6 is
   when REPORT_A =>
    TEMP := COUNT_ARRAY_A (TYPE_1, TYPE_3, TYPE_4, TYPE_2, TYPE_5);
   when REPORT_B =>
    TEMP := COUNT_ARRAY_B (TYPE_1, TYPE_3, TYPE_4, TYPE_2, TYPE_5);
   when REPORT_C =>
    TEMP := COUNT_ARRAY_C (TYPE_1, TYPE_3, TYPE_4, TYPE_2, TYPE_5);
   when REPORT_D =>
    TEMP := COUNT_ARRAY_D (TYPE_1, TYPE_3, TYPE_4, TYPE_2, TYPE_5);
   when REPORT_E =>
    TEMP := COUNT_ARRAY_E (TYPE_1, TYPE_3, TYPE_4, TYPE_2, TYPE_5);
   when REPORT_F =>
    TEMP := COUNT_ARRAY_F (TYPE_1, TYPE_3, TYPE_4, TYPE_2, TYPE_5);
  end case;

  return TEMP;

end RETRIEVE_2D_2;

--
```

```
--
function RETRIEVE_2D_3 (TYPE_1 : ORGIN;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : USAGE;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

 TEMP    : natural := 0;

begin

 case TYPE_6 is
  when REPORT_A =>
   TEMP := COUNT_ARRAY_A (TYPE_3, TYPE_2, TYPE_1, TYPE_4, TYPE_5);
  when REPORT_B =>
   TEMP := COUNT_ARRAY_B (TYPE_3, TYPE_2, TYPE_1, TYPE_4, TYPE_5);
  when REPORT_C =>
   TEMP := COUNT_ARRAY_C (TYPE_3, TYPE_2, TYPE_1, TYPE_4, TYPE_5);
  when REPORT_D =>
   TEMP := COUNT_ARRAY_D (TYPE_3, TYPE_2, TYPE_1, TYPE_4, TYPE_5);
  when REPORT_E =>
   TEMP := COUNT_ARRAY_E (TYPE_3, TYPE_2, TYPE_1, TYPE_4, TYPE_5);
  when REPORT_F =>
   TEMP := COUNT_ARRAY_F (TYPE_3, TYPE_2, TYPE_1, TYPE_4, TYPE_5);
 end case;

 return TEMP;

end RETRIEVE_2D_3;


--
--
function RETRIEVE_2D_4 (TYPE_1 : ORGIN;
                TYPE_2 : STMT_TYPE;
                TYPE_3 : HOW_PRODUCED;
                TYPE_4 : USAGE;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

 TEMP    : natural := 0;

begin

 case TYPE_6 is
  when REPORT_A =>
   TEMP := COUNT_ARRAY_A (TYPE_2, TYPE_3, TYPE_1, TYPE_4, TYPE_5);
  when REPORT_B =>
   TEMP := COUNT_ARRAY_B (TYPE_2, TYPE_3, TYPE_1, TYPE_4, TYPE_5);
  when REPORT_C =>
```

```
    TEMP := COUNT_ARRAY_C (TYPE_2, TYPE_3, TYPE_1, TYPE_4, TYPE_5);
   when REPORT_D =>
    TEMP := COUNT_ARRAY_D (TYPE_2, TYPE_3, TYPE_1, TYPE_4, TYPE_5);
   when REPORT_E =>
    TEMP := COUNT_ARRAY_E (TYPE_2, TYPE_3, TYPE_1, TYPE_4, TYPE_5);
   when REPORT_F =>
    TEMP := COUNT_ARRAY_F (TYPE_2, TYPE_3, TYPE_1, TYPE_4, TYPE_5);
  end case;

  return TEMP;

end RETRIEVE_2D_4;

--
--
function RETRIEVE_2D_5 (TYPE_1 : ORGIN;
                TYPE_2 : USAGE;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : HOW_PRODUCED;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

  TEMP    : natural := 0;

begin

  case TYPE_6 is
   when REPORT_A =>
    TEMP := COUNT_ARRAY_A (TYPE_3, TYPE_4, TYPE_1, TYPE_2, TYPE_5);
   when REPORT_B =>
    TEMP := COUNT_ARRAY_B (TYPE_3, TYPE_4, TYPE_1, TYPE_2, TYPE_5);
   when REPORT_C =>
    TEMP := COUNT_ARRAY_C (TYPE_3, TYPE_4, TYPE_1, TYPE_2, TYPE_5);
   when REPORT_D =>
    TEMP := COUNT_ARRAY_D (TYPE_3, TYPE_4, TYPE_1, TYPE_2, TYPE_5);
   when REPORT_E =>
    TEMP := COUNT_ARRAY_E (TYPE_3, TYPE_4, TYPE_1, TYPE_2, TYPE_5);
   when REPORT_F =>
    TEMP := COUNT_ARRAY_F (TYPE_3, TYPE_4, TYPE_1, TYPE_2, TYPE_5);
  end case;

  return TEMP;

end RETRIEVE_2D_5;

--
--
function RETRIEVE_2D_6 (TYPE_1 : ORGIN;
                TYPE_2 : DEVELOPMENT_STATUS;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : HOW_PRODUCED;
```

```
                TYPE_5 : USAGE;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

 TEMP    : natural := 0;

begin

 case TYPE_6 is
  when REPORT_A =>
   TEMP := COUNT_ARRAY_A (TYPE_3, TYPE_4, TYPE_1, TYPE_5, TYPE_2);
  when REPORT_B =>
   TEMP := COUNT_ARRAY_B (TYPE_3, TYPE_4, TYPE_1, TYPE_5, TYPE_2);
  when REPORT_C =>
   TEMP := COUNT_ARRAY_C (TYPE_3, TYPE_4, TYPE_1, TYPE_5, TYPE_2);
  when REPORT_D =>
   TEMP := COUNT_ARRAY_D (TYPE_3, TYPE_4, TYPE_1, TYPE_5, TYPE_2);
  when REPORT_E =>
   TEMP := COUNT_ARRAY_E (TYPE_3, TYPE_4, TYPE_1, TYPE_5, TYPE_2);
  when REPORT_F =>
   TEMP := COUNT_ARRAY_F (TYPE_3, TYPE_4, TYPE_1, TYPE_5, TYPE_2);
 end case;

 return TEMP;

end RETRIEVE_2D_6;

--
--
function RETRIEVE_2D_7 (TYPE_1 : USAGE;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : STMT_TYPE;
                TYPE_4 : ORGIN;
                TYPE_5 : DEVELOPMENT_STATUS;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

 TEMP    : natural := 0;

begin

 case TYPE_6 is
  when REPORT_A =>
   TEMP := COUNT_ARRAY_A (TYPE_3, TYPE_2, TYPE_4, TYPE_1, TYPE_5);
  when REPORT_B =>
   TEMP := COUNT_ARRAY_B (TYPE_3, TYPE_2, TYPE_4, TYPE_1, TYPE_5);
  when REPORT_C =>
   TEMP := COUNT_ARRAY_C (TYPE_3, TYPE_2, TYPE_4, TYPE_1, TYPE_5);
  when REPORT_D =>
   TEMP := COUNT_ARRAY_D (TYPE_3, TYPE_2, TYPE_4, TYPE_1, TYPE_5);
  when REPORT_E =>
   TEMP := COUNT_ARRAY_E (TYPE_3, TYPE_2, TYPE_4, TYPE_1, TYPE_5);
  when REPORT_F =>
```

```
    TEMP := COUNT_ARRAY_F (TYPE_3, TYPE_2, TYPE_4, TYPE_1, TYPE_5);
  end case;

  return TEMP;

end RETRIEVE_2D_7;

--
--
function RETRIEVE_2D_8 (TYPE_1 : DEVELOPMENT_STATUS;
                        TYPE_2 : STMT_TYPE;
                        TYPE_3 : HOW_PRODUCED;
                        TYPE_4 : ORGIN;
                        TYPE_5 : USAGE;
                        TYPE_6 : NAME_REPORT_TYPE) return natural is

  TEMP    : natural := 0;

begin

  case TYPE_6 is
    when REPORT_A =>
      TEMP := COUNT_ARRAY_A (TYPE_2, TYPE_3, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_B =>
      TEMP := COUNT_ARRAY_B (TYPE_2, TYPE_3, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_C =>
      TEMP := COUNT_ARRAY_C (TYPE_2, TYPE_3, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_D =>
      TEMP := COUNT_ARRAY_D (TYPE_2, TYPE_3, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_E =>
      TEMP := COUNT_ARRAY_E (TYPE_2, TYPE_3, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_F =>
      TEMP := COUNT_ARRAY_F (TYPE_2, TYPE_3, TYPE_4, TYPE_5, TYPE_1);
  end case;

  return TEMP;

end RETRIEVE_2D_8;

--
--
function RETRIEVE_2D_9 (TYPE_1 : DEVELOPMENT_STATUS;
                        TYPE_2 : USAGE;
                        TYPE_3 : STMT_TYPE;
                        TYPE_4 : HOW_PRODUCED;
                        TYPE_5 : ORGIN;
                        TYPE_6 : NAME_REPORT_TYPE) return natural is

  TEMP    : natural := 0;

begin
```

```
  case TYPE_6 is
    when REPORT_A =>
      TEMP := COUNT_ARRAY_A (TYPE_3, TYPE_4, TYPE_5, TYPE_2, TYPE_1);
    when REPORT_B =>
      TEMP := COUNT_ARRAY_B (TYPE_3, TYPE_4, TYPE_5, TYPE_2, TYPE_1);
    when REPORT_C =>
      TEMP := COUNT_ARRAY_C (TYPE_3, TYPE_4, TYPE_5, TYPE_2, TYPE_1);
    when REPORT_D =>
      TEMP := COUNT_ARRAY_D (TYPE_3, TYPE_4, TYPE_5, TYPE_2, TYPE_1);
    when REPORT_E =>
      TEMP := COUNT_ARRAY_E (TYPE_3, TYPE_4, TYPE_5, TYPE_2, TYPE_1);
    when REPORT_F =>
      TEMP := COUNT_ARRAY_F (TYPE_3, TYPE_4, TYPE_5, TYPE_2, TYPE_1);
  end case;

  return TEMP;

end RETRIEVE_2D_9;

--
--
function RETRIEVE_2D_10 (TYPE_1 : DEVELOPMENT_STATUS;
                         TYPE_2 : HOW_PRODUCED;
                         TYPE_3 : STMT_TYPE;
                         TYPE_4 : ORGIN;
                         TYPE_5 : USAGE;
                         TYPE_6 : NAME_REPORT_TYPE) return natural is

  TEMP    : natural := 0;

begin

  case TYPE_6 is
    when REPORT_A =>
      TEMP := COUNT_ARRAY_A (TYPE_3, TYPE_2, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_B =>
      TEMP := COUNT_ARRAY_B (TYPE_3, TYPE_2, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_C =>
      TEMP := COUNT_ARRAY_C (TYPE_3, TYPE_2, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_D =>
      TEMP := COUNT_ARRAY_D (TYPE_3, TYPE_2, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_E =>
      TEMP := COUNT_ARRAY_E (TYPE_3, TYPE_2, TYPE_4, TYPE_5, TYPE_1);
    when REPORT_F =>
      TEMP := COUNT_ARRAY_F (TYPE_3, TYPE_2, TYPE_4, TYPE_5, TYPE_1);
  end case;

  return TEMP;

end RETRIEVE_2D_10;
```

```
--
--
function RETRIEVE_3D_1 (TYPE_1 : USAGE;
                        TYPE_2 : HOW_PRODUCED;
                        TYPE_3 : ORGIN;
                        TYPE_4 : STMT_TYPE;
                        TYPE_5 : DEVELOPMENT_STATUS;
                        TYPE_6 : NAME_REPORT_TYPE) return natural is

 TEMP    : natural := 0;

begin

 case TYPE_6 is
  when REPORT_A =>
   TEMP := COUNT_ARRAY_A (TYPE_4, TYPE_2, TYPE_3, TYPE_1, TYPE_5);
  when REPORT_B =>
   TEMP := COUNT_ARRAY_B (TYPE_4, TYPE_2, TYPE_3, TYPE_1, TYPE_5);
  when REPORT_C =>
   TEMP := COUNT_ARRAY_C (TYPE_4, TYPE_2, TYPE_3, TYPE_1, TYPE_5);
  when REPORT_D =>
   TEMP := COUNT_ARRAY_D (TYPE_4, TYPE_2, TYPE_3, TYPE_1, TYPE_5);
  when REPORT_E =>
   TEMP := COUNT_ARRAY_E (TYPE_4, TYPE_2, TYPE_3, TYPE_1, TYPE_5);
  when REPORT_F =>
   TEMP := COUNT_ARRAY_F (TYPE_4, TYPE_2, TYPE_3, TYPE_1, TYPE_5);
 end case;

 return TEMP;

end RETRIEVE_3D_1;

--
--
function RETRIEVE_3D_2 (TYPE_1 : USAGE;
                        TYPE_2 : HOW_PRODUCED;
                        TYPE_3 : DEVELOPMENT_STATUS;
                        TYPE_4 : ORGIN;
                        TYPE_5 : STMT_TYPE;
                        TYPE_6 : NAME_REPORT_TYPE) return natural is

 TEMP    : natural := 0;

begin

 case TYPE_6 is
  when REPORT_A =>
   TEMP := COUNT_ARRAY_A (TYPE_5, TYPE_2, TYPE_4, TYPE_1, TYPE_3);
  when REPORT_B =>
   TEMP := COUNT_ARRAY_B (TYPE_5, TYPE_2, TYPE_4, TYPE_1, TYPE_3);
```

```
   when REPORT_C =>
    TEMP := COUNT_ARRAY_C (TYPE_5, TYPE_2, TYPE_4, TYPE_1, TYPE_3);
   when REPORT_D =>
    TEMP := COUNT_ARRAY_D (TYPE_5, TYPE_2, TYPE_4, TYPE_1, TYPE_3);
   when REPORT_E =>
    TEMP := COUNT_ARRAY_E (TYPE_5, TYPE_2, TYPE_4, TYPE_1, TYPE_3);
   when REPORT_F =>
    TEMP := COUNT_ARRAY_F (TYPE_5, TYPE_2, TYPE_4, TYPE_1, TYPE_3);
  end case;

  return TEMP;

end RETRIEVE_3D_2;

--
--
function RETRIEVE_3D_3 (TYPE_1 : DEVELOPMENT_STATUS;
                TYPE_2 : HOW_PRODUCED;
                TYPE_3 : ORGIN;
                TYPE_4 : STMT_TYPE;
                TYPE_5 : USAGE;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

  TEMP    : natural := 0;

begin

  case TYPE_6 is
   when REPORT_A =>
    TEMP := COUNT_ARRAY_A (TYPE_4, TYPE_2, TYPE_3, TYPE_5, TYPE_1);
   when REPORT_B =>
    TEMP := COUNT_ARRAY_B (TYPE_4, TYPE_2, TYPE_3, TYPE_5, TYPE_1);
   when REPORT_C =>
    TEMP := COUNT_ARRAY_C (TYPE_4, TYPE_2, TYPE_3, TYPE_5, TYPE_1);
   when REPORT_D =>
    TEMP := COUNT_ARRAY_D (TYPE_4, TYPE_2, TYPE_3, TYPE_5, TYPE_1);
   when REPORT_E =>
    TEMP := COUNT_ARRAY_E (TYPE_4, TYPE_2, TYPE_3, TYPE_5, TYPE_1);
   when REPORT_F =>
    TEMP := COUNT_ARRAY_F (TYPE_4, TYPE_2, TYPE_3, TYPE_5, TYPE_1);
  end case;

  return TEMP;

end RETRIEVE_3D_3;

--
--
function RETRIEVE_3D_4 (TYPE_1 : DEVELOPMENT_STATUS;
                TYPE_2 : USAGE;
                TYPE_3 : ORGIN;
```

```
                TYPE_4 : HOW_PRODUCED;
                TYPE_5 : STMT_TYPE;
                TYPE_6 : NAME_REPORT_TYPE) return natural is

 TEMP    : natural := 0;

begin

 case TYPE_6 is
  when REPORT_A =>
   TEMP := COUNT_ARRAY_A (TYPE_5, TYPE_4, TYPE_3, TYPE_2, TYPE_1);
  when REPORT_B =>
   TEMP := COUNT_ARRAY_B (TYPE_5, TYPE_4, TYPE_3, TYPE_2, TYPE_1);
  when REPORT_C =>
   TEMP := COUNT_ARRAY_C (TYPE_5, TYPE_4, TYPE_3, TYPE_2, TYPE_1);
  when REPORT_D =>
   TEMP := COUNT_ARRAY_D (TYPE_5, TYPE_4, TYPE_3, TYPE_2, TYPE_1);
  when REPORT_E =>
   TEMP := COUNT_ARRAY_E (TYPE_5, TYPE_4, TYPE_3, TYPE_2, TYPE_1);
  when REPORT_F =>
   TEMP := COUNT_ARRAY_F (TYPE_5, TYPE_4, TYPE_3, TYPE_2, TYPE_1);
 end case;

 return TEMP;

end RETRIEVE_3D_4;

--
--
function CHECK_2D_1 return TYPE_NUMBER_TYPE is
 T2_NUMBER : TYPE_NUMBER_TYPE := 1;
begin

 return T2_NUMBER;

end CHECK_2D_1;

--
--
function CHECK_2D_2 return TYPE_NUMBER_TYPE is
 T2_NUMBER : TYPE_NUMBER_TYPE := 2;
begin

 return T2_NUMBER;

end CHECK_2D_2;

--
--
function CHECK_2D_3 return TYPE_NUMBER_TYPE is
 T2_NUMBER : TYPE_NUMBER_TYPE := 3;
```

```
begin

 return T2_NUMBER;

end CHECK_2D_3;

--
--
function CHECK_2D_4 return TYPE_NUMBER_TYPE is
 T2_NUMBER : TYPE_NUMBER_TYPE := 4;
begin

 return T2_NUMBER;

end CHECK_2D_4;

--
--
function CHECK_3D_1 return TYPE_NUMBER_TYPE is
 T3_NUMBER : TYPE_NUMBER_TYPE := 5;
begin

 return T3_NUMBER;

end CHECK_3D_1;

--
--
function CHECK_3D_2 return TYPE_NUMBER_TYPE is
 T3_NUMBER : TYPE_NUMBER_TYPE := 2;
begin

 return T3_NUMBER;

end CHECK_3D_2;

--
--
function CHECK_3D_3 return TYPE_NUMBER_TYPE is
 T3_NUMBER : TYPE_NUMBER_TYPE := 1;
begin

 return T3_NUMBER;

end CHECK_3D_3;

--
--
function CHECK_3D_4 return TYPE_NUMBER_TYPE is
 T3_NUMBER : TYPE_NUMBER_TYPE := 4;
begin
```

```
 return T3_NUMBER;

end CHECK_3D_4;

--
--
procedure HEADING_STMT_TYPE (ROW_POSITION : positive) is

 TEMP  : integer := ROW_POSITION - 1;

begin

 if TEMP = 0 then
  put (OUT_FILE_TYPE, "Executable    ");
 elsif TEMP = 1 then
  put (OUT_FILE_TYPE, "Declarations  ");
 elsif TEMP = 2 then
  put_line (OUT_FILE_TYPE, "Compiler dir-  ");
  put (OUT_FILE_TYPE, "ectives       ");
 elsif TEMP = 3 then
  put_line (OUT_FILE_TYPE, "Comments on   ");
  put (OUT_FILE_TYPE, "their own line ");
 elsif TEMP = 4 then
  put_line (OUT_FILE_TYPE, "Comments on   ");
  put_line (OUT_FILE_TYPE, "lines with    ");
  put (OUT_FILE_TYPE, "source code   ");
 elsif TEMP = 5 then
  put_line (OUT_FILE_TYPE, "Banner and non-");
  put (OUT_FILE_TYPE, "blank spacers ");
 elsif TEMP = 6 then
  put_line (OUT_FILE_TYPE, "Blank (empty)  ");
  put (OUT_FILE_TYPE, "comments      ");
 elsif TEMP = 7 then
  put (OUT_FILE_TYPE, "Blank lines   ");
 end if;

end HEADING_STMT_TYPE;

--
--
procedure HEADING_ORGIN (ROW_POSITION : positive) is

 TEMP  : integer := ROW_POSITION - 1;

begin

 if TEMP = 0 then
  put_line (OUT_FILE_TYPE, "New Work: no  ");
  put (OUT_FILE_TYPE, "prior existence");
 elsif TEMP = 1 then
```

```
    put_line (OUT_FILE_TYPE, "A previos ver- ");
    put_line (OUT_FILE_TYPE, "sion, build,  ");
    put (OUT_FILE_TYPE, "or release    ");
   elsif TEMP = 2 then
    put (OUT_FILE_TYPE, "COTS          ");
   elsif TEMP = 3 then
    put (OUT_FILE_TYPE, "GFS           ");
   elsif TEMP = 4 then
    put (OUT_FILE_TYPE, "Another product");
   elsif TEMP = 5 then
    put_line (OUT_FILE_TYPE, "A vendor suppl-");
    put_line (OUT_FILE_TYPE, "ied language  ");
    put (OUT_FILE_TYPE, "support library");
   elsif TEMP = 6 then
    put_line (OUT_FILE_TYPE, "A vendor-suppl-");
    put_line (OUT_FILE_TYPE, "ied operating ");
    put_line (OUT_FILE_TYPE, "system or     ");
    put (OUT_FILE_TYPE, "utility       ");
   elsif TEMP = 7 then
    put_line (OUT_FILE_TYPE, "A local or mod-");
    put_line (OUT_FILE_TYPE, "ified language ");
    put_line (OUT_FILE_TYPE, "support library");
    put_line (OUT_FILE_TYPE, " or operating ");
    put (OUT_FILE_TYPE, "system        ");
   elsif TEMP = 8 then
    put_line (OUT_FILE_TYPE, "Other commer- ");
    put (OUT_FILE_TYPE, "cial library  ");
   elsif TEMP = 9 then
    put_line (OUT_FILE_TYPE, "A reuse library");
    put_line (OUT_FILE_TYPE, "(software     ");
    put_line (OUT_FILE_TYPE, "designed for  ");
    put (OUT_FILE_TYPE, "reuse         ");
   elsif TEMP = 10 then
    put_line (OUT_FILE_TYPE, "Other software ");
    put_line (OUT_FILE_TYPE, "component or  ");
    put (OUT_FILE_TYPE, "library       ");
   end if;

end HEADING_ORGIN;

--
--
procedure HEADING_USAGE (ROW_POSITION :  positive) is

  TEMP   :  integer := ROW_POSITION - 1;

begin

  if TEMP = 0 then
    put_line (OUT_FILE_TYPE, "In or as part  ");
    put_line (OUT_FILE_TYPE, "of the primary ");
```

```
   put (OUT_FILE_TYPE, "product       ");
  elsif TEMP = 1 then
   put_line (OUT_FILE_TYPE, "External to or ");
   put_line (OUT_FILE_TYPE, "in support of  ");
   put_line (OUT_FILE_TYPE, "the primary    ");
   put (OUT_FILE_TYPE, "product       ");
  end if;

end HEADING_USAGE;

--
--
procedure HEADING_DEV_STATUS (ROW_POSITION :  positive) is

 TEMP   :  integer := ROW_POSITION - 1;

begin

 if TEMP = 0 then
  put_line (OUT_FILE_TYPE, "Estimated or   ");
  put (OUT_FILE_TYPE, "planned       ");
 elsif TEMP = 1 then
  put (OUT_FILE_TYPE, "Designed      ");
 elsif TEMP = 2 then
  put (OUT_FILE_TYPE, "Coded         ");
 elsif TEMP = 3 then
  put_line (OUT_FILE_TYPE, "Unit tests com-");
  put (OUT_FILE_TYPE, "pleted        ");
 elsif TEMP = 4 then
  put_line (OUT_FILE_TYPE, "Integrated into");
  put (OUT_FILE_TYPE, "components    ");
 elsif TEMP = 5 then
  put_line (OUT_FILE_TYPE, "Test readiness ");
  put_line (OUT_FILE_TYPE, "review com-    ");
  put (OUT_FILE_TYPE, "pleted        ");
 elsif TEMP = 6 then
  put_line (OUT_FILE_TYPE, "Software (CSCI)");
  put (OUT_FILE_TYPE, "tests completed");
 elsif TEMP = 7 then
  put_line (OUT_FILE_TYPE, "System tests   ");
  put (OUT_FILE_TYPE, "completed     ");
 end if;

end HEADING_DEV_STATUS;

--
--
function FIND_PRIORITY_F (IN_STMT_TYPE  : in STMT_TYPE) return integer is

 COUNTER      :  integer := 1;
 PRIORITY_NUM  :  integer := 0;
```

```
 begin

  for F in PRIORITY_ARRAY_F'range loop
   if PRIORITY_ARRAY_F (F) = IN_STMT_TYPE then
    exit;
   else
    COUNTER := COUNTER + 1;
   end if;
  end loop;

  PRIORITY_NUM := COUNTER;
  return PRIORITY_NUM;

 end FIND_PRIORITY_F;


 --
 --
 function COUNT_TOTAL_LINES_A (IN_COUNT_TOTALS : COUNT_TOTALS_TYPE) return
natural is

  TEMP  :  natural;

 begin

  TEMP := IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL  +
       IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL   +
       IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL;

  return TEMP;

 end COUNT_TOTAL_LINES_A;


 --
 --
 function COUNT_TOTAL_LINES_B (IN_COUNT_TOTALS : in COUNT_TOTALS_TYPE)
return natural is

  TEMP  :  natural;

 begin

  TEMP := IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL      +
       IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL       +
       IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL;

  return TEMP;

 end COUNT_TOTAL_LINES_B;
```

```
--
--
function COUNT_TOTAL_LINES_C (IN_COUNT_TOTALS : in COUNT_TOTALS_TYPE)
return natural is

   TEMP  :  natural;

begin

   TEMP := IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL      +
        IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL       +
        IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL    +
        IN_COUNT_TOTALS.STMT_NUMS.CMTS_ON_OWN_TOTAL +
        IN_COUNT_TOTALS.STMT_NUMS.CMTS_W_SRC_TOTAL;

   return TEMP;

end COUNT_TOTAL_LINES_C;


--
--
function COUNT_TOTAL_LINES_D (IN_COUNT_TOTALS : in COUNT_TOTALS_TYPE)
return natural is

   TEMP  :  natural;

begin

   TEMP := IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL      +
        IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL       +
        IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL;

   return TEMP;

end COUNT_TOTAL_LINES_D;


--
--
function COUNT_TOTAL_LINES_E (IN_COUNT_TOTALS : in COUNT_TOTALS_TYPE)
return natural is

   TEMP  :  natural;

begin

   TEMP := IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL      +
        IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL       +
```

```
        IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL     +
        IN_COUNT_TOTALS.STMT_NUMS.CMTS_ON_OWN_TOTAL +
        IN_COUNT_TOTALS.STMT_NUMS.CMTS_W_SRC_TOTAL;

   return TEMP;

 end COUNT_TOTAL_LINES_E;


 --
 --
 function COUNT_TOTAL_LINES_F (IN_COUNT_TOTALS : in COUNT_TOTALS_TYPE)
return natural is

   TEMP  :  natural := 0;

 begin

   if RECORD_FLAGS_F.PANEL3.LINE_1 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL;
   end if;
   if RECORD_FLAGS_F.PANEL3.LINE_3 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL;
   end if;
   if RECORD_FLAGS_F.PANEL3.LINE_4 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL;
   end if;
   if RECORD_FLAGS_F.PANEL3.LINE_6 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.CMTS_ON_OWN_TOTAL;
   end if;
   if RECORD_FLAGS_F.PANEL3.LINE_7 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.CMTS_W_SRC_TOTAL;
   end if;
   if RECORD_FLAGS_F.PANEL3.LINE_8 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.BANNER_CMTS_TOTAL;
   end if;
   if RECORD_FLAGS_F.PANEL3.LINE_9 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.EMPTY_CMTS_TOTAL;
   end if;
   if RECORD_FLAGS_F.PANEL3.LINE_10 then
     TEMP := TEMP + IN_COUNT_TOTALS.STMT_NUMS.BLANK_LINES_TOTAL;
   end if;

   return TEMP;

 end COUNT_TOTAL_LINES_F;



 --
 --
```

```
procedure PRINT_REPORT_HEADER_1 is

 MY_TEMP  : string (1 .. 11) := (others => ' ');

begin

 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "    Report Name:   ");
 put_line (OUT_FILE_TYPE, RECORD_FLAGS.PANEL2.REPORT_HEADING (1)(1 .. 50));
 put (OUT_FILE_TYPE, "    File List used: ");
 put_line (OUT_FILE_TYPE, RECORD_FLAGS.PANEL2.IN_FILE_NAME (1)(1 .. 50));
 put (OUT_FILE_TYPE, "    Requested by:  ");
 put_line (OUT_FILE_TYPE, RECORD_FLAGS.PANEL2.REQUESTOR (1)(1 .. 50));
 new_line (OUT_FILE_TYPE);
 put_line (OUT_FILE_TYPE, "    Measured as:   Physical source lines ");
 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "    Delivered as:  ");

 MY_TEMP := GLOBAL.RECORD_FLAGS.PANEL6.DEL_OPTION (1)(1 .. 11);

 if MY_TEMP = "Delivered a" then
  put_line (OUT_FILE_TYPE, "Delivered as source");
 elsif MY_TEMP = "Delivered i" then
  put_line (OUT_FILE_TYPE, "Delivered in compiled or executable form, but not as source");
 elsif MY_TEMP = "Under confi" then
  put_line (OUT_FILE_TYPE, "Under configuration control");
 elsif MY_TEMP = "Not under c" then
  put_line (OUT_FILE_TYPE, "Not under configuration control");
 else
  put_line (OUT_FILE_TYPE, "Don't care");
 end if;

 new_line (OUT_FILE_TYPE);

end PRINT_REPORT_HEADER_1;


--
--
procedure PRINT_REPORT_HEADER_2 is

begin

 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "                                    ");
 put (OUT_FILE_TYPE, " Total    Total    Individual");
 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "                                    ");
 put (OUT_FILE_TYPE, "Includes  Excludes   totals ");
 new_line (OUT_FILE_TYPE);
```

```
end PRINT_REPORT_HEADER_2;

--
--
procedure PRINT_STMT_HEADER is

begin

 new_line (OUT_FILE_TYPE);
 put_line (OUT_FILE_TYPE, "Statement type");
 put_line (OUT_FILE_TYPE, "   When a line or statement contains more than");
 put_line (OUT_FILE_TYPE, "   one type, classify it as the type with the ");
 put_line (OUT_FILE_TYPE, "   highest precedence.");
 new_line (OUT_FILE_TYPE);

end PRINT_STMT_HEADER;

--
--
procedure PRINT_STMT_TYPE_1_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
                                REPORT_TYPE    : in integer) is
begin

 put (OUT_FILE_TYPE, " 1  Executables          Precedence => ");
 if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "1");
 else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (0)), width => 1);
 end if;
 put (OUT_FILE_TYPE, "    XXXX          ");
 put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.EXEC_TOTAL, width => 10);
 new_line (OUT_FILE_TYPE);

end PRINT_STMT_TYPE_1_IN;

--
--
procedure PRINT_STMT_TYPE_1_EX (REPORT_TYPE  :  in integer) is
begin

 put (OUT_FILE_TYPE, " 1  Executables          Precedence => ");
 if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "1");
 else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (0)), width => 1);
 end if;
 put (OUT_FILE_TYPE, "             XXXX          0");
 new_line (OUT_FILE_TYPE);

end PRINT_STMT_TYPE_1_EX;
```

```
--
--
procedure PRINT_STMT_TYPE_2 is
begin

 put_line (OUT_FILE_TYPE, " 2  Nonexecutables  ");

end PRINT_STMT_TYPE_2;

--
--
procedure PRINT_STMT_TYPE_3_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
                                REPORT_TYPE    : in integer) is
begin

 put (OUT_FILE_TYPE, " 3    Declarations                 ");
 if REPORT_TYPE > 0 then
  put (OUT_FILE_TYPE, "2");
 else
  put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (1)), width => 1);
 end if;
 put (OUT_FILE_TYPE, "    XXXX          ");
 put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.DEC_TOTAL, width => 10);
 new_line (OUT_FILE_TYPE);

end PRINT_STMT_TYPE_3_IN;

--
--
procedure PRINT_STMT_TYPE_3_EX (REPORT_TYPE  :  in integer) is
begin

 put (OUT_FILE_TYPE, " 3    Declarations                 ");
 if REPORT_TYPE > 0 then
  put (OUT_FILE_TYPE, "2");
 else
  put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (1)), width => 1);
 end if;
 put (OUT_FILE_TYPE, "          XXXX          0");
 new_line (OUT_FILE_TYPE);

end PRINT_STMT_TYPE_3_EX;


--
--
procedure PRINT_STMT_TYPE_4_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
                                REPORT_TYPE    : in integer) is
begin
```

```
 put (OUT_FILE_TYPE, " 4    Compiler Directives              ");
 if REPORT_TYPE > 0 then
  put (OUT_FILE_TYPE, "3");
 else
  put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (2)), width => 1);
 end if;
 put (OUT_FILE_TYPE, "    XXXX          ");
 put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.PRAGMA_TOTAL, width => 10);
 new_line (OUT_FILE_TYPE);

end PRINT_STMT_TYPE_4_IN;

--
--
procedure PRINT_STMT_TYPE_4_EX (REPORT_TYPE  :  in integer) is
begin

 put (OUT_FILE_TYPE, " 4    Compiler Directives              ");
 if REPORT_TYPE > 0 then
  put (OUT_FILE_TYPE, "3");
 else
  put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (2)), width => 1);
 end if;
 put (OUT_FILE_TYPE, "            XXXX          0");
 new_line (OUT_FILE_TYPE);
end PRINT_STMT_TYPE_4_EX;

--
--
procedure PRINT_STMT_TYPE_5 is
begin

 put_line (OUT_FILE_TYPE, " 5    Comments   ");

end PRINT_STMT_TYPE_5;


--
--
procedure PRINT_STMT_TYPE_6_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
                                REPORT_TYPE    : in integer) is
begin

 put (OUT_FILE_TYPE, " 6     On their own lines              ");
 if REPORT_TYPE > 0 then
  put (OUT_FILE_TYPE, "4");
 else
  put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (3)), width => 1);
 end if;
 put (OUT_FILE_TYPE, "    XXXX          ");
```

```
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.CMTS_ON_OWN_TOTAL, width
=> 10);
   new_line (OUT_FILE_TYPE);

 end PRINT_STMT_TYPE_6_IN;

 --
 --
 procedure PRINT_STMT_TYPE_6_EX (REPORT_TYPE : in integer) is
 begin

  put (OUT_FILE_TYPE, " 6     On their own lines              ");
  if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "4");
  else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (3)), width => 1);
  end if;
  put (OUT_FILE_TYPE, "          XXXX          0");
  new_line (OUT_FILE_TYPE);

 end PRINT_STMT_TYPE_6_EX;


 --
 --
 procedure PRINT_STMT_TYPE_7_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
                    REPORT_TYPE   : in integer) is
 begin

  put (OUT_FILE_TYPE, " 7     On lines with source code       ");
  if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "5");
  else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (4)), width => 1);
  end if;
  put (OUT_FILE_TYPE, "    XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.CMTS_W_SRC_TOTAL, width
=> 10);
  new_line (OUT_FILE_TYPE);

 end PRINT_STMT_TYPE_7_IN;


 --
 --
 procedure PRINT_STMT_TYPE_7_EX (REPORT_TYPE : in integer) is
 begin

  put (OUT_FILE_TYPE, " 7     On lines with source code       ");
  if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "5");
```

```
   else
    put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (4)), width => 1);
   end if;
   put (OUT_FILE_TYPE, "          XXXX          0");
   new_line (OUT_FILE_TYPE);

  end PRINT_STMT_TYPE_7_EX;



  --
  --
  procedure PRINT_STMT_TYPE_8_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
                                  REPORT_TYPE    : in integer) is
  begin

   put (OUT_FILE_TYPE, " 8     Banners and nonblank spacers     ");
   if REPORT_TYPE > 0 then
    put (OUT_FILE_TYPE, "6");
   else
    put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (5)), width => 1);
   end if;
   put (OUT_FILE_TYPE, "    XXXX          ");
   put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.BANNER_CMTS_TOTAL, width
 => 10);
   new_line (OUT_FILE_TYPE);

  end PRINT_STMT_TYPE_8_IN;


  --
  --
  procedure PRINT_STMT_TYPE_8_EX (REPORT_TYPE  :  in integer) is
  begin

   put (OUT_FILE_TYPE, " 8     Banners and nonblank spacers     ");
   if REPORT_TYPE > 0 then
    put (OUT_FILE_TYPE, "6");
   else
    put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (5)), width => 1);
   end if;
   put (OUT_FILE_TYPE, "          XXXX          0");
   new_line (OUT_FILE_TYPE);

  end PRINT_STMT_TYPE_8_EX;



  --
  --
  procedure PRINT_STMT_TYPE_9_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
```

```
                        REPORT_TYPE    : in integer) is
 begin

  put (OUT_FILE_TYPE, " 9     Blank (empty) comments          ");
  if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "7");
  else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (6)), width => 1);
  end if;
  put (OUT_FILE_TYPE, "    XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.EMPTY_CMTS_TOTAL, width
=> 10);
  new_line (OUT_FILE_TYPE);

 end PRINT_STMT_TYPE_9_IN;


 --
 --
 procedure PRINT_STMT_TYPE_9_EX (REPORT_TYPE  :  in integer) is
 begin

  put (OUT_FILE_TYPE, " 9     Blank (empty) comments          ");
  if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "7");
  else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (6)), width => 1);
  end if;
  put (OUT_FILE_TYPE, "            XXXX          0");
  new_line (OUT_FILE_TYPE);

 end PRINT_STMT_TYPE_9_EX;



 --
 --
 procedure PRINT_STMT_TYPE_10_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE;
                        REPORT_TYPE    : in integer) is
 begin

  put (OUT_FILE_TYPE, "10    Blank lines                    ");
  if REPORT_TYPE > 0 then                                    .
   put (OUT_FILE_TYPE, "8");
  else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (7)), width => 1);
  end if;
  put (OUT_FILE_TYPE, "    XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.STMT_NUMS.BLANK_LINES_TOTAL, width
=> 10);
  new_line (OUT_FILE_TYPE);
```

```
 end PRINT_STMT_TYPE_10_IN;


 --
 --
 procedure PRINT_STMT_TYPE_10_EX (REPORT_TYPE  :  in integer) is
 begin

  put (OUT_FILE_TYPE, "10   Blank lines                    ");
  if REPORT_TYPE > 0 then
   put (OUT_FILE_TYPE, "8");
  else
   put (OUT_FILE_TYPE, FIND_PRIORITY_F (STMT_TYPE'val (7)), width => 1);
  end if;
  put (OUT_FILE_TYPE, "          XXXX        0");
  new_line (OUT_FILE_TYPE);

 end PRINT_STMT_TYPE_10_EX;


 --
 --
 procedure PRINT_HOW_PRODUCED is
 begin

  new_line (OUT_FILE_TYPE);
  put_line (OUT_FILE_TYPE, "How Produced");
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED;

 --
 --
 procedure PRINT_HOW_PRODUCED_1_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 1  Programmed                          XXXX         ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.PRODUCED_NUMS.PROGRAMMED_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);
 end PRINT_HOW_PRODUCED_1_IN;

 --
 --
 procedure PRINT_HOW_PRODUCED_1_EX is
 begin
  put (OUT_FILE_TYPE, " 1  Programmed                                  XXXX         0");
  new_line (OUT_FILE_TYPE);
 end PRINT_HOW_PRODUCED_1_EX;
 --
```

```
--
 procedure PRINT_HOW_PRODUCED_2_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 2  Generated with source code generators      XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.PRODUCED_NUMS.GENERATED_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);
 end PRINT_HOW_PRODUCED_2_IN;

 --
 --
 procedure PRINT_HOW_PRODUCED_2_EX is
 begin
  put (OUT_FILE_TYPE, " 2  Generated with source code generators              XXXX          0");
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_2_EX;

 --
 --
 procedure PRINT_HOW_PRODUCED_3_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 3  Converted with automated translators      XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.PRODUCED_NUMS.CONVERTED_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_3_IN;

 --
 --
 procedure PRINT_HOW_PRODUCED_3_EX is
 begin
  put (OUT_FILE_TYPE, " 3  Converted with automated translators              XXXX          0");
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_3_EX;
 --
 --
 procedure PRINT_HOW_PRODUCED_4_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 4  Copied or reused without change          XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.PRODUCED_NUMS.COPIED_TOTAL, width
=> 10);
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_4_IN;
```

```
 --
 --
 procedure PRINT_HOW_PRODUCED_4_EX is
 begin
  put (OUT_FILE_TYPE, " 4  Copied or reused without change                XXXX          0");
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_4_EX;

 --
 --
 procedure PRINT_HOW_PRODUCED_5_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 5  Modified                              XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.PRODUCED_NUMS.MODIFIED_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_5_IN;

 --
 --
 procedure PRINT_HOW_PRODUCED_5_EX is
 begin
  put (OUT_FILE_TYPE, " 5  Modified                                  XXXX          0");
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_5_EX;


 --
 --
 procedure PRINT_HOW_PRODUCED_6_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 6  Removed                               XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.PRODUCED_NUMS.REMOVED_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);

 end PRINT_HOW_PRODUCED_6_IN;


 --
 --
 procedure PRINT_HOW_PRODUCED_6_EX is
 begin
  put (OUT_FILE_TYPE, " 6  Removed                                   XXXX          0");
  new_line (OUT_FILE_TYPE);
 end PRINT_HOW_PRODUCED_6_EX;
```

```
--
--
procedure PRINT_ORGIN is
begin
  new_line (OUT_FILE_TYPE);
  put_line (OUT_FILE_TYPE, "Orgin");
  new_line (OUT_FILE_TYPE);
end PRINT_ORGIN;

--
--
procedure PRINT_ORGIN_1_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
begin
  put (OUT_FILE_TYPE, " 1  New Work:  no prior existence          XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.ORGIN_NUMS.NEW_WORK_TOTAL, width
=> 10);
  new_line (OUT_FILE_TYPE);
end PRINT_ORGIN_1_IN;

--
--
procedure PRINT_ORGIN_1_EX is
begin
  put (OUT_FILE_TYPE, " 1  New Work:  no prior existence                XXXX");
  put_line (OUT_FILE_TYPE, "          0");
end PRINT_ORGIN_1_EX;

--
--
procedure PRINT_ORGIN_2 is
begin
  put_line (OUT_FILE_TYPE, " 2  Prior work:  taken or adapted from ");
end PRINT_ORGIN_2;

--
--
procedure PRINT_ORGIN_3_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
begin
  put (OUT_FILE_TYPE, " 3    A previous version, build, or release    XXXX          ");
  put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.ORGIN_NUMS.PREVIOUS_VERSION_TOTAL, width => 10);
  new_line (OUT_FILE_TYPE);
end PRINT_ORGIN_3_IN;

--
--
procedure PRINT_ORGIN_3_EX is
begin
  put (OUT_FILE_TYPE, " 3    A previous version, build, or release          XXXX");
  put_line (OUT_FILE_TYPE, "          0");
```

```
 end PRINT_ORGIN_3_EX;

 --
 --
 procedure PRINT_ORGIN_4_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
  put_line (OUT_FILE_TYPE, " 4   Commercial, off the shelf software");
  put (OUT_FILE_TYPE, "     COTS), other than libraries          XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.ORGIN_NUMS.COTS_TOTAL, width => 10);
  new_line (OUT_FILE_TYPE);
 end PRINT_ORGIN_4_IN;

 --
 --
 procedure PRINT_ORGIN_4_EX is
 begin
  put_line (OUT_FILE_TYPE, " 4   Commercial, off the shelf software");
  put (OUT_FILE_TYPE, "     COTS), other than libraries                XXXX");
  put_line (OUT_FILE_TYPE, "          0");
 end PRINT_ORGIN_4_EX;

 --
 --
 procedure PRINT_ORGIN_5_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
  put_line (OUT_FILE_TYPE, " 5   Government furnished software (GFS),");
  put (OUT_FILE_TYPE, "     other than reuse libraries          XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.ORGIN_NUMS.GFS_TOTAL, width => 10);
  new_line (OUT_FILE_TYPE);
 end PRINT_ORGIN_5_IN;

 --
 --
 procedure PRINT_ORGIN_5_EX is
 begin
  put (OUT_FILE_TYPE, " 5   Government furnished software (GFS),");
  put (OUT_FILE_TYPE, "     other than reuse libraries                XXXX");
  put_line (OUT_FILE_TYPE, "          ");
 end PRINT_ORGIN_5_EX;

 --
 --
 procedure PRINT_ORGIN_6_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 6   Another product                XXXX          ");
  put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.ORGIN_NUMS.ANNOTHER_PRODUCT_TOTAL, width => 10);
  new_line (OUT_FILE_TYPE);
 end PRINT_ORGIN_6_IN;

 --
```

```
--
procedure PRINT_ORGIN_6_EX is
begin
  put (OUT_FILE_TYPE, " 6    Another product                              XXXX");
  put_line (OUT_FILE_TYPE, "          0");
end  PRINT_ORGIN_6_EX;

--
--
procedure PRINT_ORGIN_7_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
begin
  put_line (OUT_FILE_TYPE, " 7    A vendor-supplied language support");
  put (OUT_FILE_TYPE, "      library (unmodified)                 XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.ORGIN_NUMS.VS_SPT_LIB_TOTAL, width =>
10);
  new_line (OUT_FILE_TYPE);
end  PRINT_ORGIN_7_IN;

--
--
procedure PRINT_ORGIN_7_EX is
begin
  put_line (OUT_FILE_TYPE, " 7    A vendor-supplied language support");
  put (OUT_FILE_TYPE, "      library (unmodified)                         XXXX");
  put_line (OUT_FILE_TYPE, "          0");
end  PRINT_ORGIN_7_EX;

--
--
procedure PRINT_ORGIN_8_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
begin
  put_line (OUT_FILE_TYPE, " 8    A vedor-supplied operating system or");
  put (OUT_FILE_TYPE, "      utility (unmodified)                 XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.ORGIN_NUMS.VS_SPT_OS_TOTAL, width =>
10);
  new_line (OUT_FILE_TYPE);
end  PRINT_ORGIN_8_IN;

--
--
procedure PRINT_ORGIN_8_EX is
begin
  put_line (OUT_FILE_TYPE, " 8    A vedor-supplied operating system or");
  put (OUT_FILE_TYPE, "      utility (unmodified)                         XXXX");
  put_line (OUT_FILE_TYPE, "          0");
end  PRINT_ORGIN_8_EX;

--
--
procedure PRINT_ORGIN_9_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
begin
```

```
   put_line (OUT_FILE_TYPE, " 9    A local or modified language support");
   put (OUT_FILE_TYPE, "     library or operating system          XXXX          ");
   put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.ORGIN_NUMS.LOCAL_SUPPLIED_LIB_TOTAL, width => 10);
   new_line (OUT_FILE_TYPE);
 end  PRINT_ORGIN_9_IN;

 --
 --
 procedure PRINT_ORGIN_9_EX is
 begin
   put_line (OUT_FILE_TYPE, " 9    A local or modified language support");
   put (OUT_FILE_TYPE, "     library or operating system                XXXX");
   put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_ORGIN_9_EX;

 --
 --
 procedure PRINT_ORGIN_10_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
   put (OUT_FILE_TYPE, "10   Other commercial library          XXXX          ");
   put (OUT_FILE_TYPE, IN_COUNT_TOTAL.ORGIN_NUMS.COMMERCIAL_LIB_TOTAL,
width => 10);
   new_line (OUT_FILE_TYPE);
 end  PRINT_ORGIN_10_IN;

 --
 --
 procedure PRINT_ORGIN_10_EX is
 begin
   put (OUT_FILE_TYPE, "10   Other commercial library                XXXX");
   put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_ORGIN_10_EX;

 --
 --
 procedure PRINT_ORGIN_11_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
   put_line (OUT_FILE_TYPE, "11   A reuse library (software designed");
   put (OUT_FILE_TYPE, "     for reuse)                    XXXX          ");
   put (OUT_FILE_TYPE, IN_COUNT_TOTAL.ORGIN_NUMS.REUSE_LIB_TOTAL, width =>
10);
   new_line (OUT_FILE_TYPE);
 end  PRINT_ORGIN_11_IN;

 --
 --
 procedure PRINT_ORGIN_11_EX is
 begin
   put_line (OUT_FILE_TYPE, "11   A reuse library (software designed");
   put (OUT_FILE_TYPE, "     for reuse)                          XXXX");
```

```
  put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_ORGIN_11_EX;

 --
 --
 procedure PRINT_ORGIN_12_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, "12   Other software component or library      XXXX          ");
  put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.ORGIN_NUMS.OTHER_COMPONENT_TOTAL, width => 10);
  new_line (OUT_FILE_TYPE);
 end  PRINT_ORGIN_12_IN;

 --
 --
 procedure PRINT_ORGIN_12_EX is
 begin
  put (OUT_FILE_TYPE, "12   Other software component or library              XXXX");
  put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_ORGIN_12_EX;

 --
 --
 procedure PRINT_USAGE is
 begin
  new_line (OUT_FILE_TYPE);
  put_line (OUT_FILE_TYPE, "Usage");
  new_line (OUT_FILE_TYPE);
 end PRINT_USAGE;

 --
 --
 procedure PRINT_USAGE_1_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 1  In or as part of the primary product      XXXX          ");
  put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.USAGE_NUMS.PRIMARY_PRODUCT_TOTAL, width => 10);
  new_line (OUT_FILE_TYPE);
 end PRINT_USAGE_1_IN;

 --
 --
 procedure PRINT_USAGE_1_EX is
 begin
  put (OUT_FILE_TYPE, " 1  In or as part of the primary product              XXXX          0");
  new_line (OUT_FILE_TYPE);
 end PRINT_USAGE_1_EX;


 --
 --
```

```
 procedure PRINT_USAGE_2_IN (IN_COUNT_TOTAL : in COUNT_TOTALS_TYPE) is
 begin
  put_line (OUT_FILE_TYPE, " 2  External to or in support of the");
  put (OUT_FILE_TYPE, "    primary product                    XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.USAGE_NUMS.EXTERNAL_TOTAL, width =>
10);
  new_line (OUT_FILE_TYPE);
 end PRINT_USAGE_2_IN;

 --
 --
 procedure PRINT_USAGE_2_EX is
 begin
  put_line (OUT_FILE_TYPE, " 2  External to or in support of the");
  put (OUT_FILE_TYPE, "    primary product                              XXXX          0");
  new_line (OUT_FILE_TYPE);
 end PRINT_USAGE_2_EX;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS is
 begin
  new_line (OUT_FILE_TYPE);
  put_line (OUT_FILE_TYPE, "Development Status");
  new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_1_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 1  Estimated or planned                XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.DEVELOPED_NUMS.ESTIMATED_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS_1_IN;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_1_EX is
 begin
  put (OUT_FILE_TYPE, " 1  Estimated or planned                          XXXX");
  put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_DEVELOPMENT_STATUS_1_EX;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_2_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
```

```
   put (OUT_FILE_TYPE, " 2  Designed                    XXXX          ");
   put (OUT_FILE_TYPE, IN_COUNT_TOTAL.DEVELOPED_NUMS.DESIGNED_TOTAL,
width => 10);
   new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS_2_IN;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_2_EX is
 begin
   put (OUT_FILE_TYPE, " 2  Designed                          XXXX");
   put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_DEVELOPMENT_STATUS_2_EX;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_3_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
   put (OUT_FILE_TYPE, " 3  Coded                       XXXX          ");
   put (OUT_FILE_TYPE, IN_COUNT_TOTAL.DEVELOPED_NUMS.CODED_TOTAL, width
=> 10);
   new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS_3_IN;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_3_EX is
 begin
   put (OUT_FILE_TYPE, " 3  Coded                          XXXX");
   put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_DEVELOPMENT_STATUS_3_EX;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_4_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
   put (OUT_FILE_TYPE, " 4  Unit tests completed            XXXX          ");
   put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.DEVELOPED_NUMS.UNIT_TEST_DONE_TOTAL, width => 10);
   new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS_4_IN;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_4_EX is
 begin
   put (OUT_FILE_TYPE, " 4  Unit tests completed                XXXX");
   put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_DEVELOPMENT_STATUS_4_EX;
```

```
 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_5_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 5  Integrated into components              XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.DEVELOPED_NUMS.INTEGRATED_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS_5_IN;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_5_EX is
 begin
  put (OUT_FILE_TYPE, " 5  Integrated into components                    XXXX");
  put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_DEVELOPMENT_STATUS_5_EX;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_6_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 6  Test readiness review completed          XXXX          ");
  put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.DEVELOPED_NUMS.TEST_READINESS_REVIEW_TOTAL, width =>
10);
  new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS_6_IN;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_6_EX is
 begin
  put (OUT_FILE_TYPE, " 6  Test readiness review completed                XXXX");
  put_line (OUT_FILE_TYPE, "          0");
 end  PRINT_DEVELOPMENT_STATUS_6_EX;

 --
 --
 procedure PRINT_DEVELOPMENT_STATUS_7_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
 begin
  put (OUT_FILE_TYPE, " 7  Software (CSCI) tests completed          XXXX          ");
  put (OUT_FILE_TYPE,
IN_COUNT_TOTAL.DEVELOPED_NUMS.CSCI_COMPLETED_TOTAL, width => 10);
  new_line (OUT_FILE_TYPE);
 end  PRINT_DEVELOPMENT_STATUS_7_IN;
```

```
--
--
procedure PRINT_DEVELOPMENT_STATUS_7_EX is
begin
  put (OUT_FILE_TYPE, " 7  Software (CSCI) tests completed                XXXX");
  put_line (OUT_FILE_TYPE, "          0");
end  PRINT_DEVELOPMENT_STATUS_7_EX;

--
--
procedure PRINT_DEVELOPMENT_STATUS_8_IN (IN_COUNT_TOTAL : in
COUNT_TOTALS_TYPE) is
begin
  put (OUT_FILE_TYPE, " 8  System tests completed                XXXX          ");
  put (OUT_FILE_TYPE, IN_COUNT_TOTAL.DEVELOPED_NUMS.SYSTEM_TEST_TOTAL,
width => 10);
  new_line (OUT_FILE_TYPE);
end  PRINT_DEVELOPMENT_STATUS_8_IN;

--
--
procedure PRINT_DEVELOPMENT_STATUS_8_EX is
begin
  put (OUT_FILE_TYPE, " 8  System tests completed                XXXX");
  put_line (OUT_FILE_TYPE, "          0");
end  PRINT_DEVELOPMENT_STATUS_8_EX;

--
--
procedure PRINT_DATA_ARRAY_F is
begin

  -- check for 3D arrays
  if  RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
      RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
      RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY    then
    RETRIEVE_1_2D.INTERFACE_3D_MAT (REPORT_F);
  end if;
  if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
       RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
       RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY   then
    RETRIEVE_2_2D.INTERFACE_3D_MAT (REPORT_F);
  end if;
  if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
       RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
       RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY    then
    RETRIEVE_8_2D.INTERFACE_3D_MAT (REPORT_F);
  end if;
  if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
       RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY and
       RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
```

```
 RETRIEVE_5_2D.INTERFACE_3D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
 RETRIEVE_6_2D.INTERFACE_3D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY    then
 RETRIEVE_9_2D.INTERFACE_3D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
 RETRIEVE_1_3D.INTERFACE_3D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
 RETRIEVE_3_3D.INTERFACE_3D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY    then
 RETRIEVE_2_3D.INTERFACE_3D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
 RETRIEVE_4_3D.INTERFACE_3D_MAT (REPORT_F);
end if;

-- check for 2D arrays
if  RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
    RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY    then
 RETRIEVE_1_2D.INTERFACE_2D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
 RETRIEVE_4_2D.INTERFACE_2D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY    then
 RETRIEVE_2_2D.INTERFACE_2D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL3.DEF_DATA_ARRAY and
     RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY   then
 RETRIEVE_8_2D.INTERFACE_2D_MAT (REPORT_F);
end if;
if   RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
```

```
    RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
  RETRIEVE_3_2D.INTERFACE_2D_MAT (REPORT_F);
 end if;
 if   RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
    RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY    then
  RETRIEVE_7_2D.INTERFACE_2D_MAT (REPORT_F);
 end if;
 if   RECORD_FLAGS_F.PANEL4.DEF_DATA_ARRAY and
    RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY   then
  RETRIEVE_10_2D.INTERFACE_2D_MAT (REPORT_F);
 end if;
 if   RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY and
    RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY   then
  RETRIEVE_5_2D.INTERFACE_2D_MAT (REPORT_F);
 end if;
 if   RECORD_FLAGS_F.PANEL5.DEF_DATA_ARRAY and
    RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY    then
  RETRIEVE_6_2D.INTERFACE_2D_MAT (REPORT_F);
 end if;
 if   RECORD_FLAGS_F.PANEL9.DEF_DATA_ARRAY and
    RECORD_FLAGS_F.PANEL6.DEF_DATA_ARRAY   then
  RETRIEVE_9_2D.INTERFACE_2D_MAT (REPORT_F);
 end if;

end PRINT_DATA_ARRAY_F;


--
--
function COUNT_STMT_TYPE (S           : in STMT_TYPE;
                IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT  :  integer  := 0;

begin

 for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
  for O in ORGIN'FIRST .. ORGIN'LAST  loop
   for U in USAGE'FIRST .. USAGE'LAST loop
    for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
     TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
    end loop;
   end loop;
  end loop;
 end loop;

 return TEMP_COUNT;

end COUNT_STMT_TYPE;
```

```
--
--
function COUNT_HOW_PRODUCED (H           : in HOW_PRODUCED;
                             IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT  : integer := 0;

begin

 for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
  for O in ORGIN'FIRST .. ORGIN'LAST loop
   for U in USAGE'FIRST .. USAGE'LAST loop
    for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
     TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
    end loop;
   end loop;
  end loop;
 end loop;

 return TEMP_COUNT;

end COUNT_HOW_PRODUCED;


function COUNT_ORGIN (O           : in ORGIN;
                      IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT  : integer := 0;

begin

 for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
  for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
   for U in USAGE'FIRST .. USAGE'LAST loop
    for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
     TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
    end loop;
   end loop;
  end loop;
 end loop;

 return TEMP_COUNT;

end COUNT_ORGIN;



function COUNT_USAGE (U           : in USAGE;
                      IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

 TEMP_COUNT  : integer := 0;
```

```
begin

  for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
   for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
    for O in ORGIN'FIRST .. ORGIN'LAST loop
     for D in DEVELOPMENT_STATUS'FIRST .. DEVELOPMENT_STATUS'LAST loop
      TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
     end loop;
    end loop;
   end loop;
  end loop;

  return TEMP_COUNT;

end COUNT_USAGE;


function COUNT_DEVELOPMENT_STATUS (D           : in DEVELOPMENT_STATUS;
                                   IN_COUNT_ARRAY : in COUNT_ARRAY_TYPE) return integer is

  TEMP_COUNT  :  integer  := 0;

begin

  for S in STMT_TYPE'FIRST .. STMT_TYPE'LAST loop
   for H in HOW_PRODUCED'FIRST .. HOW_PRODUCED'LAST loop
    for O in ORGIN'FIRST .. ORGIN'LAST loop
     for U in USAGE'FIRST .. USAGE'LAST loop
      TEMP_COUNT := TEMP_COUNT + IN_COUNT_ARRAY (S, H, O, U, D);
     end loop;
    end loop;
   end loop;
  end loop;

  return TEMP_COUNT;

end COUNT_DEVELOPMENT_STATUS;

--
--
procedure COUNT_ATTRIBUTE_ONE (IN_RECORD_FLAGS : in    FLAGS;
                               IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                               IN_ARRAY        : in    COUNT_ARRAY_TYPE) is
begin

  if IN_RECORD_FLAGS.PANEL3.line_1 then
   IN_COUNT_TOTALS.STMT_NUMS.EXEC_TOTAL := COUNT_STMT_TYPE
(STMT_TYPE'val (0), IN_ARRAY);
  end if;
  if IN_RECORD_FLAGS.PANEL3.line_3 then
```

```
    IN_COUNT_TOTALS.STMT_NUMS.DEC_TOTAL := COUNT_STMT_TYPE
(STMT_TYPE'val (1), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL3.line_4 then
    IN_COUNT_TOTALS.STMT_NUMS.PRAGMA_TOTAL := COUNT_STMT_TYPE
(STMT_TYPE'val (2), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL3.line_6 then
    IN_COUNT_TOTALS.STMT_NUMS.CMTS_ON_OWN_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (3), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL3.line_7 then
    IN_COUNT_TOTALS.STMT_NUMS.CMTS_W_SRC_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (4), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL3.line_8 then
    IN_COUNT_TOTALS.STMT_NUMS.BANNER_CMTS_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (5), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL3.line_9 then
    IN_COUNT_TOTALS.STMT_NUMS.EMPTY_CMTS_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (6), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL3.line_10 then
    IN_COUNT_TOTALS.STMT_NUMS.BLANK_LINES_TOTAL :=
          COUNT_STMT_TYPE (STMT_TYPE'val (7), IN_ARRAY);
   end if;

 end COUNT_ATTRIBUTE_ONE;

 --
 --
 procedure COUNT_ATTRIBUTE_TWO (IN_RECORD_FLAGS : in    FLAGS;
                    IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                    IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

   if IN_RECORD_FLAGS.PANEL4.line_1 then
    IN_COUNT_TOTALS.PRODUCED_NUMS.PROGRAMMED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (0), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL4.line_2 then
    IN_COUNT_TOTALS.PRODUCED_NUMS.GENERATED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (1), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL4.line_3 then
    IN_COUNT_TOTALS.PRODUCED_NUMS.CONVERTED_TOTAL :=
          COUNT_HOW_PRODUCED (HOW_PRODUCED'val (2), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL4.line_4 then
    IN_COUNT_TOTALS.PRODUCED_NUMS.COPIED_TOTAL :=
```

```
            COUNT_HOW_PRODUCED (HOW_PRODUCED'val (3), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL4.line_5 then
    IN_COUNT_TOTALS.PRODUCED_NUMS.MODIFIED_TOTAL :=
            COUNT_HOW_PRODUCED (HOW_PRODUCED'val (4), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL4.line_6 then
    IN_COUNT_TOTALS.PRODUCED_NUMS.REMOVED_TOTAL :=
            COUNT_HOW_PRODUCED (HOW_PRODUCED'val (5), IN_ARRAY);
   end if;

 end COUNT_ATTRIBUTE_TWO;



 procedure COUNT_ATTRIBUTE_THREE (IN_RECORD_FLAGS : in    FLAGS;
                      IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                      IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

   if IN_RECORD_FLAGS.PANEL5.line_1 then
    IN_COUNT_TOTALS.ORGIN_NUMS.NEW_WORK_TOTAL := COUNT_ORGIN
(ORGIN'val (0), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_3 then
    IN_COUNT_TOTALS.ORGIN_NUMS.PREVIOUS_VERSION_TOTAL :=
            COUNT_ORGIN (ORGIN'val (1), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_4 then
    IN_COUNT_TOTALS.ORGIN_NUMS.COTS_TOTAL := COUNT_ORGIN (ORGIN'val (2),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_5 then
    IN_COUNT_TOTALS.ORGIN_NUMS.GFS_TOTAL := COUNT_ORGIN (ORGIN'val (3),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_6 then
    IN_COUNT_TOTALS.ORGIN_NUMS.ANNOTHER_PRODUCT_TOTAL :=
            COUNT_ORGIN (ORGIN'val (4), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_7 then
    IN_COUNT_TOTALS.ORGIN_NUMS.VS_SPT_LIB_TOTAL := COUNT_ORGIN
(ORGIN'val (5), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_8 then
    IN_COUNT_TOTALS.ORGIN_NUMS.VS_SPT_OS_TOTAL := COUNT_ORGIN
(ORGIN'val (6), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_9 then
    IN_COUNT_TOTALS.ORGIN_NUMS.LOCAL_SUPPLIED_LIB_TOTAL :=
            COUNT_ORGIN (ORGIN'val (7), IN_ARRAY);
```

```
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_10 then
    IN_COUNT_TOTALS.ORGIN_NUMS.COMMERCIAL_LIB_TOTAL :=
          COUNT_ORGIN (ORGIN'val (8), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_11 then
    IN_COUNT_TOTALS.ORGIN_NUMS.REUSE_LIB_TOTAL := COUNT_ORGIN
(ORGIN'val (9), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL5.line_12 then
    IN_COUNT_TOTALS.ORGIN_NUMS.OTHER_COMPONENT_TOTAL :=
          COUNT_ORGIN (ORGIN'val (10), IN_ARRAY);
   end if;


 end COUNT_ATTRIBUTE_THREE;

 --
 --
 procedure COUNT_ATTRIBUTE_FOUR (IN_RECORD_FLAGS : in    FLAGS;
                    IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                    IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

   if IN_RECORD_FLAGS.PANEL6.line_1 then
    IN_COUNT_TOTALS.USAGE_NUMS.PRIMARY_PRODUCT_TOTAL :=
          COUNT_USAGE (USAGE'val (0), IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL6.line_2 then
    IN_COUNT_TOTALS.USAGE_NUMS.EXTERNAL_TOTAL := COUNT_USAGE
(USAGE'val (1), IN_ARRAY);
   end if;

 end COUNT_ATTRIBUTE_FOUR;

 --
 --
 procedure COUNT_ATTRIBUTE_FIVE (IN_RECORD_FLAGS : in    FLAGS;
                    IN_COUNT_TOTALS : in out COUNT_TOTALS_TYPE;
                    IN_ARRAY      : in    COUNT_ARRAY_TYPE) is
 begin

   if IN_RECORD_FLAGS.PANEL9.line_1 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.ESTIMATED_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (0),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL9.line_2 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.DESIGNED_TOTAL :=
          COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (1),
IN_ARRAY);
```

```
   end if;
   if IN_RECORD_FLAGS.PANEL9.line_3 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.CODED_TOTAL :=
            COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (2),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL9.line_4 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.UNIT_TEST_DONE_TOTAL :=
            COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (3),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL9.line_5 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.INTEGRATED_TOTAL :=
            COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (4),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL9.line_6 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.TEST_READINESS_REVIEW_TOTAL :=
            COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (5),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL9.line_7 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.CSCI_COMPLETED_TOTAL :=
            COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (6),
IN_ARRAY);
   end if;
   if IN_RECORD_FLAGS.PANEL9.line_8 then
    IN_COUNT_TOTALS.DEVELOPED_NUMS.SYSTEM_TEST_TOTAL :=
            COUNT_DEVELOPMENT_STATUS (DEVELOPMENT_STATUS'val (7),
IN_ARRAY);
   end if;

 end COUNT_ATTRIBUTE_FIVE;


 --
 --
 procedure REPORT_A (COUNT_TOTALS_A : in out COUNT_TOTALS_TYPE) is

  REPORT_A_FLAG     : integer := 1;
  ESTIMATED_TOTAL   : DEVELOPMENT_STATUS := DEVELOPMENT_STATUS'val(0);

 begin

  new_line (OUT_FILE_TYPE, 2);
  put_line (OUT_FILE_TYPE, "                    REPORT A");

  PRINT_REPORT_HEADER_1;

  new_line (OUT_FILE_TYPE);
  put (OUT_FILE_TYPE, "      Counted:   ");
```

```
   put (OUT_FILE_TYPE, COUNT_TOTAL_LINES_A (COUNT_TOTALS_A));
   new_line (OUT_FILE_TYPE);
   put (OUT_FILE_TYPE, "      Estimated: ");
   put (OUT_FILE_TYPE, CNT_EST (ESTIMATED_TOTAL, COUNT_ARRAY_A));

   new_line (OUT_FILE_TYPE);
   PRINT_REPORT_HEADER_2;

   PRINT_STMT_HEADER;
   PRINT_STMT_TYPE_1_IN (COUNT_TOTALS_A, REPORT_A_FLAG);
   PRINT_STMT_TYPE_2;
   PRINT_STMT_TYPE_3_IN (COUNT_TOTALS_A, REPORT_A_FLAG);
   PRINT_STMT_TYPE_4_IN (COUNT_TOTALS_A, REPORT_A_FLAG);
   PRINT_STMT_TYPE_5;
   PRINT_STMT_TYPE_6_EX (REPORT_A_FLAG);
   PRINT_STMT_TYPE_7_EX (REPORT_A_FLAG);
   PRINT_STMT_TYPE_8_EX (REPORT_A_FLAG);
   PRINT_STMT_TYPE_9_EX (REPORT_A_FLAG);
   PRINT_STMT_TYPE_10_EX (REPORT_A_FLAG);

   PRINT_HOW_PRODUCED;
   PRINT_HOW_PRODUCED_1_IN (COUNT_TOTALS_A);
   PRINT_HOW_PRODUCED_2_IN (COUNT_TOTALS_A);
   PRINT_HOW_PRODUCED_3_IN (COUNT_TOTALS_A);
   PRINT_HOW_PRODUCED_4_IN (COUNT_TOTALS_A);
   PRINT_HOW_PRODUCED_5_IN (COUNT_TOTALS_A);
   PRINT_HOW_PRODUCED_6_EX;

   PRINT_ORGIN;
   PRINT_ORGIN_1_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_2;
   PRINT_ORGIN_3_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_4_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_5_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_6_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_7_EX;
   PRINT_ORGIN_8_EX;
   PRINT_ORGIN_9_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_10_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_11_IN (COUNT_TOTALS_A);
   PRINT_ORGIN_12_IN (COUNT_TOTALS_A);

--   PRINT_REPORT_HEADER_2;

   PRINT_USAGE;
   PRINT_USAGE_1_IN (COUNT_TOTALS_A);
   PRINT_USAGE_2_EX;

   PRINT_DEVELOPMENT_STATUS;
   PRINT_DEVELOPMENT_STATUS_1_EX;
   PRINT_DEVELOPMENT_STATUS_2_EX;
```

```
 PRINT_DEVELOPMENT_STATUS_3_EX;
 PRINT_DEVELOPMENT_STATUS_4_EX;
 PRINT_DEVELOPMENT_STATUS_5_EX;
 PRINT_DEVELOPMENT_STATUS_6_EX;
 PRINT_DEVELOPMENT_STATUS_7_EX;
 PRINT_DEVELOPMENT_STATUS_8_IN (COUNT_TOTALS_A);

end REPORT_A;


--
--
procedure REPORT_B (COUNT_TOTALS_B : in COUNT_TOTALS_TYPE) is

 REPORT_B_FLAG     : integer := 1;
 ESTIMATED_TOTAL   : DEVELOPMENT_STATUS := DEVELOPMENT_STATUS'val(0);

begin

 new_page (OUT_FILE_TYPE);

 new_line (OUT_FILE_TYPE, 2);
 put_line (OUT_FILE_TYPE, "                    REPORT B");
 new_line (OUT_FILE_TYPE);
 PRINT_REPORT_HEADER_1;

 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "      Counted:   ");
 put (OUT_FILE_TYPE, COUNT_TOTAL_LINES_B (COUNT_TOTALS_B));
 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "      Estimated: ");
 put (OUT_FILE_TYPE, CNT_EST (ESTIMATED_TOTAL, COUNT_ARRAY_B));
 new_line (OUT_FILE_TYPE);

 PRINT_REPORT_HEADER_2;

 PRINT_STMT_HEADER;
 PRINT_STMT_TYPE_1_IN (COUNT_TOTALS_B, REPORT_B_FLAG);
 PRINT_STMT_TYPE_2;
 PRINT_STMT_TYPE_3_IN (COUNT_TOTALS_B, REPORT_B_FLAG);
 PRINT_STMT_TYPE_4_IN (COUNT_TOTALS_B, REPORT_B_FLAG);
 PRINT_STMT_TYPE_5;
 PRINT_STMT_TYPE_6_EX (REPORT_B_FLAG);
 PRINT_STMT_TYPE_7_EX (REPORT_B_FLAG);
 PRINT_STMT_TYPE_8_EX (REPORT_B_FLAG);
 PRINT_STMT_TYPE_9_EX (REPORT_B_FLAG);
 PRINT_STMT_TYPE_10_EX (REPORT_B_FLAG);

 PRINT_HOW_PRODUCED;
 PRINT_HOW_PRODUCED_1_IN (COUNT_TOTALS_B);
 PRINT_HOW_PRODUCED_2_IN (COUNT_TOTALS_B);
```

```
   PRINT_HOW_PRODUCED_3_IN (COUNT_TOTALS_B);
   PRINT_HOW_PRODUCED_4_IN (COUNT_TOTALS_B);
   PRINT_HOW_PRODUCED_5_IN (COUNT_TOTALS_B);
   PRINT_HOW_PRODUCED_6_IN (COUNT_TOTALS_B);

   PRINT_ORGIN;
   PRINT_ORGIN_1_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_2;
   PRINT_ORGIN_3_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_4_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_5_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_6_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_7_EX;
   PRINT_ORGIN_8_EX;
   PRINT_ORGIN_9_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_10_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_11_IN (COUNT_TOTALS_B);
   PRINT_ORGIN_12_IN (COUNT_TOTALS_B);

--   PRINT_REPORT_HEADER_2;

   PRINT_USAGE;
   PRINT_USAGE_1_IN (COUNT_TOTALS_B);
   PRINT_USAGE_2_EX;

   PRINT_DEVELOPMENT_STATUS;
   PRINT_DEVELOPMENT_STATUS_1_EX;
   PRINT_DEVELOPMENT_STATUS_2_EX;
   PRINT_DEVELOPMENT_STATUS_3_IN (COUNT_TOTALS_B);
   PRINT_DEVELOPMENT_STATUS_4_IN (COUNT_TOTALS_B);
   PRINT_DEVELOPMENT_STATUS_5_IN (COUNT_TOTALS_B);
   PRINT_DEVELOPMENT_STATUS_6_IN (COUNT_TOTALS_B);
   PRINT_DEVELOPMENT_STATUS_7_IN (COUNT_TOTALS_B);
   PRINT_DEVELOPMENT_STATUS_8_IN (COUNT_TOTALS_B);

   -- Using generic package to print out two dimensional array
   -- of Development_status and How_produced
   RETRIEVE_10_2D.INTERFACE_2D_MAT (REPORT_B);

 end REPORT_B;

 --
 --
 procedure REPORT_C (COUNT_TOTALS_C : in COUNT_TOTALS_TYPE) is

   REPORT_C_FLAG     : integer := 1;
   ESTIMATED_TOTAL   : DEVELOPMENT_STATUS := DEVELOPMENT_STATUS'val(0);

 begin

   new_page (OUT_FILE_TYPE);
```

```
   new_line (OUT_FILE_TYPE, 2);
   put_line (OUT_FILE_TYPE, "                    REPORT C");
   new_line (OUT_FILE_TYPE);
   PRINT_REPORT_HEADER_1;

   new_line (OUT_FILE_TYPE);
   put (OUT_FILE_TYPE, "      Counted:  ");
   put (OUT_FILE_TYPE, COUNT_TOTAL_LINES_C (COUNT_TOTALS_C));
   new_line (OUT_FILE_TYPE);
   put (OUT_FILE_TYPE, "      Estimated: ");
   put (OUT_FILE_TYPE, CNT_EST (ESTIMATED_TOTAL, COUNT_ARRAY_C));
   new_line (OUT_FILE_TYPE);
   PRINT_REPORT_HEADER_2;

   PRINT_STMT_HEADER;
   PRINT_STMT_TYPE_1_IN (COUNT_TOTALS_C, REPORT_C_FLAG);
   PRINT_STMT_TYPE_2;
   PRINT_STMT_TYPE_3_IN (COUNT_TOTALS_C, REPORT_C_FLAG);
   PRINT_STMT_TYPE_4_IN (COUNT_TOTALS_C, REPORT_C_FLAG);
   PRINT_STMT_TYPE_5;
   PRINT_STMT_TYPE_6_IN (COUNT_TOTALS_C, REPORT_C_FLAG);
   PRINT_STMT_TYPE_7_IN (COUNT_TOTALS_C, REPORT_C_FLAG);
   PRINT_STMT_TYPE_8_EX (REPORT_C_FLAG);
   PRINT_STMT_TYPE_9_EX (REPORT_C_FLAG);
   PRINT_STMT_TYPE_10_EX (REPORT_C_FLAG);

   PRINT_HOW_PRODUCED;
   PRINT_HOW_PRODUCED_1_IN (COUNT_TOTALS_C);
   PRINT_HOW_PRODUCED_2_IN (COUNT_TOTALS_C);
   PRINT_HOW_PRODUCED_3_IN (COUNT_TOTALS_C);
   PRINT_HOW_PRODUCED_4_IN (COUNT_TOTALS_C);
   PRINT_HOW_PRODUCED_5_IN (COUNT_TOTALS_C);
   PRINT_HOW_PRODUCED_6_IN (COUNT_TOTALS_C);

   PRINT_ORGIN;
   PRINT_ORGIN_1_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_2;
   PRINT_ORGIN_3_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_4_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_5_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_6_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_7_EX;
   PRINT_ORGIN_8_EX;
   PRINT_ORGIN_9_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_10_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_11_IN (COUNT_TOTALS_C);
   PRINT_ORGIN_12_IN (COUNT_TOTALS_C);

--   PRINT_REPORT_HEADER_2;
```

```
  PRINT_USAGE;
  PRINT_USAGE_1_IN (COUNT_TOTALS_C);
  PRINT_USAGE_2_EX;

  PRINT_DEVELOPMENT_STATUS;
  PRINT_DEVELOPMENT_STATUS_1_EX;
  PRINT_DEVELOPMENT_STATUS_2_EX;
  PRINT_DEVELOPMENT_STATUS_3_EX;
  PRINT_DEVELOPMENT_STATUS_4_EX;
  PRINT_DEVELOPMENT_STATUS_5_EX;
  PRINT_DEVELOPMENT_STATUS_6_EX;
  PRINT_DEVELOPMENT_STATUS_7_EX;
  PRINT_DEVELOPMENT_STATUS_8_IN (COUNT_TOTALS_C);

  -- Using generic package to print out two dimensional array
  -- of Statement_type and How_produced
  RETRIEVE_1_2D.INTERFACE_2D_MAT (REPORT_C);

end REPORT_C;

--
--
procedure REPORT_D (COUNT_TOTALS_D : in COUNT_TOTALS_TYPE) is

  REPORT_D_FLAG     : integer := 1;
  ESTIMATED_TOTAL   : DEVELOPMENT_STATUS := DEVELOPMENT_STATUS'val(0);

begin

  new_page (OUT_FILE_TYPE);

  new_line (OUT_FILE_TYPE, 2);
  put_line (OUT_FILE_TYPE, "                    REPORT D");
  new_line (OUT_FILE_TYPE);
  PRINT_REPORT_HEADER_1;

  new_line (OUT_FILE_TYPE);
  put (OUT_FILE_TYPE, "      Counted:  ");
  put (OUT_FILE_TYPE, COUNT_TOTAL_LINES_D (COUNT_TOTALS_D));
  new_line (OUT_FILE_TYPE);
  put (OUT_FILE_TYPE, "      Estimated:  ");
  put (OUT_FILE_TYPE, CNT_EST (ESTIMATED_TOTAL, COUNT_ARRAY_D));

  new_line (OUT_FILE_TYPE);
  PRINT_REPORT_HEADER_2;

  PRINT_STMT_HEADER;
  PRINT_STMT_TYPE_1_IN (COUNT_TOTALS_D, REPORT_D_FLAG);
  PRINT_STMT_TYPE_2;
  PRINT_STMT_TYPE_3_IN (COUNT_TOTALS_D, REPORT_D_FLAG);
  PRINT_STMT_TYPE_4_IN (COUNT_TOTALS_D, REPORT_D_FLAG);
```

```
   PRINT_STMT_TYPE_5;
   PRINT_STMT_TYPE_6_EX (REPORT_D_FLAG);
   PRINT_STMT_TYPE_7_EX (REPORT_D_FLAG);
   PRINT_STMT_TYPE_8_EX (REPORT_D_FLAG);
   PRINT_STMT_TYPE_9_EX (REPORT_D_FLAG);
   PRINT_STMT_TYPE_10_EX (REPORT_D_FLAG);

   PRINT_HOW_PRODUCED;
   PRINT_HOW_PRODUCED_1_IN (COUNT_TOTALS_D);
   PRINT_HOW_PRODUCED_2_IN (COUNT_TOTALS_D);
   PRINT_HOW_PRODUCED_3_IN (COUNT_TOTALS_D);
   PRINT_HOW_PRODUCED_4_IN (COUNT_TOTALS_D);
   PRINT_HOW_PRODUCED_5_IN (COUNT_TOTALS_D);
   PRINT_HOW_PRODUCED_6_IN (COUNT_TOTALS_D);

   PRINT_ORGIN;
   PRINT_ORGIN_1_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_2;
   PRINT_ORGIN_3_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_4_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_5_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_6_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_7_EX;
   PRINT_ORGIN_8_EX;
   PRINT_ORGIN_9_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_10_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_11_IN (COUNT_TOTALS_D);
   PRINT_ORGIN_12_IN (COUNT_TOTALS_D);

--   PRINT_REPORT_HEADER_2;

   PRINT_USAGE;
   PRINT_USAGE_1_IN (COUNT_TOTALS_D);
   PRINT_USAGE_2_EX;

   PRINT_DEVELOPMENT_STATUS;
   PRINT_DEVELOPMENT_STATUS_1_EX;
   PRINT_DEVELOPMENT_STATUS_2_EX;
   PRINT_DEVELOPMENT_STATUS_3_EX;
   PRINT_DEVELOPMENT_STATUS_4_EX;
   PRINT_DEVELOPMENT_STATUS_5_EX;
   PRINT_DEVELOPMENT_STATUS_6_EX;
   PRINT_DEVELOPMENT_STATUS_7_EX;
   PRINT_DEVELOPMENT_STATUS_8_IN (COUNT_TOTALS_D);

   -- Using generic package to print out two dimensional array
   -- of Orgin and How_produced
   RETRIEVE_3_2D.INTERFACE_2D_MAT (REPORT_D);

 end REPORT_D;
```

```
--
--
procedure REPORT_E (COUNT_TOTALS_E : in COUNT_TOTALS_TYPE) is

 REPORT_E_FLAG     : integer := 1;
 ESTIMATED_TOTAL   : DEVELOPMENT_STATUS := DEVELOPMENT_STATUS'val(0);

begin

 new_page (OUT_FILE_TYPE);

 new_line (OUT_FILE_TYPE, 2);
 put_line (OUT_FILE_TYPE, "                    REPORT E");
 new_line (OUT_FILE_TYPE);
 PRINT_REPORT_HEADER_1;
 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "      Counted:   ");
 put (OUT_FILE_TYPE, COUNT_TOTAL_LINES_E (COUNT_TOTALS_E));
 new_line (OUT_FILE_TYPE);
 put (OUT_FILE_TYPE, "      Estimated: ");
 put (OUT_FILE_TYPE, CNT_EST (ESTIMATED_TOTAL, COUNT_ARRAY_E));
 new_line (OUT_FILE_TYPE);
 PRINT_REPORT_HEADER_2;
 PRINT_STMT_HEADER;
 PRINT_STMT_TYPE_1_IN (COUNT_TOTALS_E, REPORT_E_FLAG);
 PRINT_STMT_TYPE_2;
 PRINT_STMT_TYPE_3_IN (COUNT_TOTALS_E, REPORT_E_FLAG);
 PRINT_STMT_TYPE_4_IN (COUNT_TOTALS_E, REPORT_E_FLAG);
 PRINT_STMT_TYPE_5;
 PRINT_STMT_TYPE_6_IN (COUNT_TOTALS_E, REPORT_E_FLAG);
 PRINT_STMT_TYPE_7_IN (COUNT_TOTALS_E, REPORT_E_FLAG);
 PRINT_STMT_TYPE_8_EX (REPORT_E_FLAG);
 PRINT_STMT_TYPE_9_EX (REPORT_E_FLAG);
 PRINT_STMT_TYPE_10_EX (REPORT_E_FLAG);

 PRINT_HOW_PRODUCED;
 PRINT_HOW_PRODUCED_1_IN (COUNT_TOTALS_E);
 PRINT_HOW_PRODUCED_2_IN (COUNT_TOTALS_E);
 PRINT_HOW_PRODUCED_3_IN (COUNT_TOTALS_E);
 PRINT_HOW_PRODUCED_4_IN (COUNT_TOTALS_E);
 PRINT_HOW_PRODUCED_5_IN (COUNT_TOTALS_E);
 PRINT_HOW_PRODUCED_6_IN (COUNT_TOTALS_E);

 PRINT_ORGIN;
 PRINT_ORGIN_1_IN (COUNT_TOTALS_E);
 PRINT_ORGIN_2;
 PRINT_ORGIN_3_IN (COUNT_TOTALS_E);
 PRINT_ORGIN_4_IN (COUNT_TOTALS_E);
 PRINT_ORGIN_5_IN (COUNT_TOTALS_E);
 PRINT_ORGIN_6_IN (COUNT_TOTALS_E);
 PRINT_ORGIN_7_EX;
```

```
   PRINT_ORGIN_8_EX;
   PRINT_ORGIN_9_IN (COUNT_TOTALS_E);
   PRINT_ORGIN_10_IN (COUNT_TOTALS_E);
   PRINT_ORGIN_11_IN (COUNT_TOTALS_E);
   PRINT_ORGIN_12_IN (COUNT_TOTALS_E);

--   PRINT_REPORT_HEADER_2;

   PRINT_USAGE;
   PRINT_USAGE_1_IN (COUNT_TOTALS_E);
   PRINT_USAGE_2_EX;

   PRINT_DEVELOPMENT_STATUS;
   PRINT_DEVELOPMENT_STATUS_1_EX;
   PRINT_DEVELOPMENT_STATUS_2_EX;
   PRINT_DEVELOPMENT_STATUS_3_EX;
   PRINT_DEVELOPMENT_STATUS_4_EX;
   PRINT_DEVELOPMENT_STATUS_5_EX;
   PRINT_DEVELOPMENT_STATUS_6_EX;
   PRINT_DEVELOPMENT_STATUS_7_EX;
   PRINT_DEVELOPMENT_STATUS_8_IN (COUNT_TOTALS_E);

   RETRIEVE_1_2D.INTERFACE_3D_MAT (REPORT_E);

 end REPORT_E;


 --
 --
 procedure REPORT_F (COUNT_TOTALS_F : in COUNT_TOTALS_TYPE) is

   REPORT_F_FLAG     : integer := -1;
   ESTIMATED_TOTAL   : DEVELOPMENT_STATUS := DEVELOPMENT_STATUS'val(0);

 begin

   new_page (OUT_FILE_TYPE);

   new_line (OUT_FILE_TYPE, 2);
   put_line (OUT_FILE_TYPE, "                    REPORT F");
   new_line (OUT_FILE_TYPE);

--   Print_out_panel_3_settings;

   PRINT_REPORT_HEADER_1;
   new_line (OUT_FILE_TYPE);
   put (OUT_FILE_TYPE, "      Counted:  ");
   put (OUT_FILE_TYPE, COUNT_TOTAL_LINES_F (COUNT_TOTALS_F));
   new_line (OUT_FILE_TYPE);
   put (OUT_FILE_TYPE, "      Estimated:  ");
   put (OUT_FILE_TYPE, CNT_EST (ESTIMATED_TOTAL, COUNT_ARRAY_F));
```

```
new_line (OUT_FILE_TYPE);
PRINT_REPORT_HEADER_2;
PRINT_STMT_HEADER;
if RECORD_FLAGS_F.PANEL3.LINE_1  then
 PRINT_STMT_TYPE_1_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_1_EX (REPORT_F_FLAG);
end if;

PRINT_STMT_TYPE_2;

if RECORD_FLAGS_F.PANEL3.LINE_3  then
 PRINT_STMT_TYPE_3_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_3_EX (REPORT_F_FLAG);
end if;
if RECORD_FLAGS_F.PANEL3.LINE_4 then
 PRINT_STMT_TYPE_4_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_4_EX (REPORT_F_FLAG);
end if;

PRINT_STMT_TYPE_5;

if RECORD_FLAGS_F.PANEL3.LINE_6 then
 PRINT_STMT_TYPE_6_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_6_EX (REPORT_F_FLAG);
end if;
if RECORD_FLAGS_F.PANEL3.LINE_7 then
 PRINT_STMT_TYPE_7_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_7_EX (REPORT_F_FLAG);
end if;
if RECORD_FLAGS_F.PANEL3.LINE_8 then
 PRINT_STMT_TYPE_8_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_8_EX (REPORT_F_FLAG);
end if;
if RECORD_FLAGS_F.PANEL3.LINE_9 then
 PRINT_STMT_TYPE_9_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_9_EX (REPORT_F_FLAG);
end if;
if RECORD_FLAGS_F.PANEL3.LINE_10 then
 PRINT_STMT_TYPE_10_IN (COUNT_TOTALS_F, REPORT_F_FLAG);
else
 PRINT_STMT_TYPE_10_EX (REPORT_F_FLAG);
end if;

PRINT_HOW_PRODUCED;
```

```
if RECORD_FLAGS_F.PANEL4.LINE_1 then
 PRINT_HOW_PRODUCED_1_IN (COUNT_TOTALS_F);
else
 PRINT_HOW_PRODUCED_1_EX;
end if;
if RECORD_FLAGS_F.PANEL4.LINE_2 then
 PRINT_HOW_PRODUCED_2_IN (COUNT_TOTALS_F);
else
 PRINT_HOW_PRODUCED_2_EX;
end if;
if RECORD_FLAGS_F.PANEL4.LINE_3 then
 PRINT_HOW_PRODUCED_3_IN (COUNT_TOTALS_F);
else
 PRINT_HOW_PRODUCED_3_EX;
end if;
if RECORD_FLAGS_F.PANEL4.LINE_4 then
 PRINT_HOW_PRODUCED_4_IN (COUNT_TOTALS_F);
else
 PRINT_HOW_PRODUCED_4_EX;
end if;
if RECORD_FLAGS_F.PANEL4.LINE_5  then
 PRINT_HOW_PRODUCED_5_IN (COUNT_TOTALS_F);
else
 PRINT_HOW_PRODUCED_5_EX;
end if;
if RECORD_FLAGS_F.PANEL4.LINE_6 then
 PRINT_HOW_PRODUCED_6_IN (COUNT_TOTALS_F);
else
 PRINT_HOW_PRODUCED_6_EX;
end if;

PRINT_ORGIN;

if RECORD_FLAGS_F.PANEL5.LINE_1 then
 PRINT_ORGIN_1_IN (COUNT_TOTALS_F);
else
 PRINT_ORGIN_1_EX;
end if;

PRINT_ORGIN_2;

if RECORD_FLAGS_F.PANEL5.LINE_3 then
 PRINT_ORGIN_3_IN (COUNT_TOTALS_F);
else
 PRINT_ORGIN_3_EX;
end if;
if RECORD_FLAGS_F.PANEL5.LINE_4 then
 PRINT_ORGIN_4_IN (COUNT_TOTALS_F);
else
 PRINT_ORGIN_4_EX;
```

```
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_5 then
     PRINT_ORGIN_5_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_5_EX;
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_6 then
     PRINT_ORGIN_6_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_6_EX;
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_7 then
     PRINT_ORGIN_7_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_7_EX;
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_8 then
     PRINT_ORGIN_8_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_8_EX;
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_9 then
     PRINT_ORGIN_9_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_9_EX;
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_10 then
     PRINT_ORGIN_10_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_10_EX;
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_11 then
     PRINT_ORGIN_11_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_11_EX;
    end if;
    if RECORD_FLAGS_F.PANEL5.LINE_12 then
     PRINT_ORGIN_12_IN (COUNT_TOTALS_F);
    else
     PRINT_ORGIN_12_EX;
    end if;

--   PRINT_REPORT_HEADER_2;

    PRINT_USAGE;

    if RECORD_FLAGS_F.PANEL6.LINE_1 then
     PRINT_USAGE_1_IN (COUNT_TOTALS_F);
    else
     PRINT_USAGE_1_EX;
    end if;
```

```
 if RECORD_FLAGS_F.PANEL6.LINE_2 then
  PRINT_USAGE_2_IN (COUNT_TOTALS_F);
 else
  PRINT_USAGE_2_EX;
 end if;

 PRINT_DEVELOPMENT_STATUS;

 if RECORD_FLAGS_F.PANEL9.LINE_1 then
  PRINT_DEVELOPMENT_STATUS_1_IN (COUNT_TOTALS_F);
 else
  PRINT_DEVELOPMENT_STATUS_1_EX;
 end if;
 if RECORD_FLAGS_F.PANEL9.LINE_2 then
  PRINT_DEVELOPMENT_STATUS_2_IN (COUNT_TOTALS_F);
 else
  PRINT_DEVELOPMENT_STATUS_2_EX;
 end if;
 if RECORD_FLAGS_F.PANEL9.LINE_3 then
  PRINT_DEVELOPMENT_STATUS_3_IN (COUNT_TOTALS_F);
 else
  PRINT_DEVELOPMENT_STATUS_3_EX;
 end if;
 if RECORD_FLAGS_F.PANEL9.LINE_4 then
  PRINT_DEVELOPMENT_STATUS_4_IN (COUNT_TOTALS_F);
 else
  PRINT_DEVELOPMENT_STATUS_4_EX;
 end if;
 if RECORD_FLAGS_F.PANEL9.LINE_5 then
  PRINT_DEVELOPMENT_STATUS_5_IN (COUNT_TOTALS_F);
 else
  PRINT_DEVELOPMENT_STATUS_5_EX;
 end if;
 if RECORD_FLAGS_F.PANEL9.LINE_6  then
  PRINT_DEVELOPMENT_STATUS_6_IN (COUNT_TOTALS_F);
 else
  PRINT_DEVELOPMENT_STATUS_6_EX;
 end if;
 if RECORD_FLAGS_F.PANEL9.LINE_7 then
  PRINT_DEVELOPMENT_STATUS_7_EX;
 else
  PRINT_DEVELOPMENT_STATUS_7_EX;
 end if;
 if RECORD_FLAGS_F.PANEL9.LINE_8 then
  PRINT_DEVELOPMENT_STATUS_8_IN (COUNT_TOTALS_F);
 else
  PRINT_DEVELOPMENT_STATUS_8_EX;
 end if;

PRINT_DATA_ARRAY_F;
```

```
end REPORT_F;


--
--
procedure DETERMINE_WHICH_REPORT is

begin

  if  RECORD_FLAGS.PANEL2.REPORT_A then
    COUNT_ATTRIBUTE_ONE (RECORD_FLAGS_A, COUNT_TOTALS_A,
COUNT_ARRAY_A);
    COUNT_ATTRIBUTE_TWO (RECORD_FLAGS_A, COUNT_TOTALS_A,
COUNT_ARRAY_A);
    COUNT_ATTRIBUTE_THREE (RECORD_FLAGS_A, COUNT_TOTALS_A,
COUNT_ARRAY_A);
    COUNT_ATTRIBUTE_FOUR (RECORD_FLAGS_A, COUNT_TOTALS_A,
COUNT_ARRAY_A);
    COUNT_ATTRIBUTE_FIVE (RECORD_FLAGS_A, COUNT_TOTALS_A,
COUNT_ARRAY_A);
    REPORT_A (COUNT_TOTALS_A);
  end if;

  if  RECORD_FLAGS.PANEL2.REPORT_B then
    COUNT_ATTRIBUTE_ONE (RECORD_FLAGS_B, COUNT_TOTALS_B,
COUNT_ARRAY_B);
    COUNT_ATTRIBUTE_TWO (RECORD_FLAGS_B, COUNT_TOTALS_B,
COUNT_ARRAY_B);
    COUNT_ATTRIBUTE_THREE (RECORD_FLAGS_B, COUNT_TOTALS_B,
COUNT_ARRAY_B);
    COUNT_ATTRIBUTE_FOUR (RECORD_FLAGS_B, COUNT_TOTALS_B,
COUNT_ARRAY_B);
    COUNT_ATTRIBUTE_FIVE (RECORD_FLAGS_B, COUNT_TOTALS_B,
COUNT_ARRAY_B);
    REPORT_B (COUNT_TOTALS_B);
  end if;

  if  RECORD_FLAGS.PANEL2.REPORT_C then
    COUNT_ATTRIBUTE_ONE (RECORD_FLAGS_C, COUNT_TOTALS_C,
COUNT_ARRAY_C);
    COUNT_ATTRIBUTE_TWO (RECORD_FLAGS_C, COUNT_TOTALS_C,
COUNT_ARRAY_C);
    COUNT_ATTRIBUTE_THREE (RECORD_FLAGS_C, COUNT_TOTALS_C,
COUNT_ARRAY_C);
    COUNT_ATTRIBUTE_FOUR (RECORD_FLAGS_C, COUNT_TOTALS_C,
COUNT_ARRAY_C);
    COUNT_ATTRIBUTE_FIVE (RECORD_FLAGS_C, COUNT_TOTALS_C,
COUNT_ARRAY_C);
    REPORT_C (COUNT_TOTALS_C);
  end if;
```

```
   if  RECORD_FLAGS.PANEL2.REPORT_D then
     COUNT_ATTRIBUTE_ONE (RECORD_FLAGS_D, COUNT_TOTALS_D,
COUNT_ARRAY_D);
     COUNT_ATTRIBUTE_TWO (RECORD_FLAGS_D, COUNT_TOTALS_D,
COUNT_ARRAY_D);
     COUNT_ATTRIBUTE_THREE (RECORD_FLAGS_D, COUNT_TOTALS_D,
COUNT_ARRAY_D);
     COUNT_ATTRIBUTE_FOUR (RECORD_FLAGS_D, COUNT_TOTALS_D,
COUNT_ARRAY_D);
     COUNT_ATTRIBUTE_FIVE (RECORD_FLAGS_D, COUNT_TOTALS_D,
COUNT_ARRAY_D);
     REPORT_D (COUNT_TOTALS_D);
   end if;

   if  RECORD_FLAGS.PANEL2.REPORT_E then
     COUNT_ATTRIBUTE_ONE (RECORD_FLAGS_E, COUNT_TOTALS_E,
COUNT_ARRAY_E);
     COUNT_ATTRIBUTE_TWO (RECORD_FLAGS_E, COUNT_TOTALS_E,
COUNT_ARRAY_E);
     COUNT_ATTRIBUTE_THREE (RECORD_FLAGS_E, COUNT_TOTALS_E,
COUNT_ARRAY_E);
     COUNT_ATTRIBUTE_FOUR (RECORD_FLAGS_E, COUNT_TOTALS_E,
COUNT_ARRAY_E);
     COUNT_ATTRIBUTE_FIVE (RECORD_FLAGS_E, COUNT_TOTALS_E,
COUNT_ARRAY_E);
     REPORT_E (COUNT_TOTALS_E);
   end if;

   if  RECORD_FLAGS.PANEL2.REPORT_F then
     COUNT_ATTRIBUTE_ONE (RECORD_FLAGS_F, COUNT_TOTALS_F,
COUNT_ARRAY_F);
     COUNT_ATTRIBUTE_TWO (RECORD_FLAGS_F, COUNT_TOTALS_F,
COUNT_ARRAY_F);
     COUNT_ATTRIBUTE_THREE (RECORD_FLAGS_F, COUNT_TOTALS_F,
COUNT_ARRAY_F);
     COUNT_ATTRIBUTE_FOUR (RECORD_FLAGS_F, COUNT_TOTALS_F,
COUNT_ARRAY_F);
     COUNT_ATTRIBUTE_FIVE (RECORD_FLAGS_F, COUNT_TOTALS_F,
COUNT_ARRAY_F);
     REPORT_F (COUNT_TOTALS_F);
   end if;


 end DETERMINE_WHICH_REPORT;


end REPORT_PACKAGE;
```

GENERIC PACKAGE SPEC AND BODY

```
--*_Programmed
with text_io,
   GLOBAL;
use  text_io,
   GLOBAL;

generic
 type FIRST_TYPE is  (<>);
 type SECOND_TYPE is (<>);
 type THIRD_TYPE is  (<>);
 type FOURTH_TYPE is (<>);
 type FIFTH_TYPE is  (<>);
 type REPORT_TYPE is (<>);
 type T_NUMBER_TYPE is range <>;
 with function RETRIEVE (TYPE1 : FIRST_TYPE;
                         TYPE2 : SECOND_TYPE;
                         TYPE3 : THIRD_TYPE;
                         TYPE4 : FOURTH_TYPE;
                         TYPE5 : FIFTH_TYPE;
                         TYPE6 : REPORT_TYPE) return natural;
 with function CHECK_TYPE_2 return T_NUMBER_TYPE;
 with function CHECK_TYPE_3 return T_NUMBER_TYPE;
 with procedure PRINT_ROW_HEADING (ROW_NUM : positive);

--
--
package GENERIC_COUNTS is

 package INTEGER_IN_OUT is new integer_io (integer);
 use INTEGER_IN_OUT;

 package ENUMERATION_IN_OUT is new ENUMERATION_IO (STMT_TYPE);
 use ENUMERATION_IN_OUT;

 --
 --
 procedure INTERFACE_2D_MAT (T6 : in REPORT_TYPE);

 --
 --
 procedure INTERFACE_3D_MAT (T6 : in REPORT_TYPE);


end GENERIC_COUNTS;

--
--
```

```
package body GENERIC_COUNTS is

 --
 --
 procedure HOW_PRODUCED_HEADING (IN_NUM : positive := 20) is
  TEMP   : natural  := IN_NUM - 1;
 begin

  if IN_NUM > 15 then
   new_line (OUT_FILE_TYPE);
   put_line (OUT_FILE_TYPE, "          Programmed Generated Converted   Copied  Modified
Removed");
   new_line (OUT_FILE_TYPE, 2);
  else
   if TEMP = 0 then
    put (OUT_FILE_TYPE, "          How produced.Programmed");
   elsif TEMP = 1 then
    put (OUT_FILE_TYPE, "          How produced.Generated with source code generators");
   elsif TEMP = 2 then
    put (OUT_FILE_TYPE, "          How produced.Converted with automated translators");
   elsif TEMP = 3 then
    put (OUT_FILE_TYPE, "          How produced.Copied or reused without change");
   elsif TEMP = 4 then
    put (OUT_FILE_TYPE, "          How produced.Modified");
   elsif TEMP = 5 then
    put (OUT_FILE_TYPE, "          How produced.Removed");
   end if;
  end if;

 end HOW_PRODUCED_HEADING;

 --
 --
 procedure STMT_TYPE_HEADING (IN_NUM : positive := 20) is
  TEMP   : natural  := IN_NUM - 1;
 begin

  if IN_NUM > 15 then
   new_line (OUT_FILE_TYPE);
   put_line (OUT_FILE_TYPE, "            Exec     Dec    Pragma   Cmmts Cmmts_w_other
Banner   Empty   Blank");
   new_line (OUT_FILE_TYPE, 2);
  else
   if TEMP = 0 then
    put (OUT_FILE_TYPE, "          Statement type.Executable");
   elsif TEMP = 1 then
    put (OUT_FILE_TYPE, "          Statement type.Declarations");
   elsif TEMP = 2 then
    put (OUT_FILE_TYPE, "          Statement type.Compiler Directives");
   elsif TEMP = 3 then
    put (OUT_FILE_TYPE, "          Statement type.Comments on their own lines");
```

```
    elsif TEMP = 4 then
      put (OUT_FILE_TYPE, "              Statement type.Comments on lines with source code");
    elsif TEMP = 5 then
      put (OUT_FILE_TYPE, "              Statement type.Banners and nonblank spacers");
    elsif TEMP = 6 then
      put (OUT_FILE_TYPE, "              Statement type.Blank (empty) comments");
    elsif TEMP = 7 then
      put (OUT_FILE_TYPE, "              Statement type.Blank lines");
    end if;
  end if;

end STMT_TYPE_HEADING;


--
--
procedure USAGE_HEADING (IN_NUM : positive := 20) is
  TEMP   : natural  := IN_NUM - 1;
begin

  if IN_NUM > 15 then
    new_line (OUT_FILE_TYPE);
    put_line (OUT_FILE_TYPE, "              Primary   External");
    new_line (OUT_FILE_TYPE, 2);
  else
    if TEMP = 0 then
      put (OUT_FILE_TYPE, "              Usage.In or as part of the primary product");
    elsif TEMP = 1 then
      put (OUT_FILE_TYPE, "              Usage.External to or in support of the primary product");
    end if;
  end if;

end USAGE_HEADING;

--
--
procedure DEVELOPMENT_STATUS_HEADING (IN_NUM : positive := 20) is

  TEMP   : natural  := IN_NUM - 1;

begin

  if IN_NUM > 15 then
    new_line (OUT_FILE_TYPE);
    put_line (OUT_FILE_TYPE, "              Est    Designed   Coded   Unit test Integrated
Readiness   CSCI   System Tests");
    new_line (OUT_FILE_TYPE, 2);
  else
    if TEMP = 0 then
      put (OUT_FILE_TYPE, "              Development Status.Estimated or planned");
    elsif TEMP = 1 then
```

```
    put (OUT_FILE_TYPE, "          Development Status.Designed");
   elsif TEMP = 2 then
    put (OUT_FILE_TYPE, "          Development Status.Coded");
   elsif TEMP = 3 then
    put (OUT_FILE_TYPE, "          Development Status.Unit tests completed");
   elsif TEMP = 4 then
    put (OUT_FILE_TYPE, "          Development Status.Integrated into components");
   elsif TEMP = 5 then
    put (OUT_FILE_TYPE, "          Development Status.Test readiness completed");
   elsif TEMP = 6 then
    put (OUT_FILE_TYPE, "          Development Status.Software (CSCI) tests completed");
   elsif TEMP = 7 then
    put (OUT_FILE_TYPE, "          Development Status.System tests completed");
   end if;
  end if;

end DEVELOPMENT_STATUS_HEADING;

--
--
procedure ORGIN_HEADING (IN_NUM : positive := 20) is

 TEMP   : natural := IN_NUM - 1;

begin

 if TEMP = 0 then
  put (OUT_FILE_TYPE, "          Orgin.New work");
 elsif TEMP = 1 then
  put (OUT_FILE_TYPE, "          Orgin.Previos version");
 elsif TEMP = 2 then
  put (OUT_FILE_TYPE, "          Orgin.COTS");
 elsif TEMP = 3 then
  put (OUT_FILE_TYPE, "          Orgin.GFS");
 elsif TEMP = 4 then
  put (OUT_FILE_TYPE, "          Orgin.Annother product");
 elsif TEMP = 5 then
  put (OUT_FILE_TYPE, "          Orgin.A vendor supplied language support library");
 elsif TEMP = 6 then
  put (OUT_FILE_TYPE, "          Orgin.A vendor supplied operating system or utility");
 elsif TEMP = 7 then
  put (OUT_FILE_TYPE, "          Orgin.A local or modified language support library");
 elsif TEMP = 8 then
  put (OUT_FILE_TYPE, "          Orgin.Other commercial library");
 elsif TEMP = 9 then
  put (OUT_FILE_TYPE, "          Orgin.A reuse library");
 elsif TEMP = 10 then
  put (OUT_FILE_TYPE, "          Orgin.Other software component or library");
 end if;

 new_line (OUT_FILE_TYPE);
```

```
end ORGIN_HEADING;

--
--
function GEN_2D_MAT (FIX_1 : in FIRST_TYPE;
                     FIX_2 : in SECOND_TYPE;
                     IN_T6 : in REPORT_TYPE) return natural is

 SUM    : natural := 0;

begin

 for T3 in THIRD_TYPE'first .. THIRD_TYPE'last loop

  for T4 in FOURTH_TYPE'first .. FOURTH_TYPE'last loop

   for T5 in FIFTH_TYPE'first .. FIFTH_TYPE'last loop

    SUM := SUM + RETRIEVE (FIX_1, FIX_2, T3, T4, T5, IN_T6);

   end loop;

  end loop;

 end loop;

 return SUM;

end GEN_2D_MAT;

--
--
function GEN_3D_MAT (FIX_1 : in FIRST_TYPE;
                     FIX_2 : in SECOND_TYPE;
                     FIX_3 : in THIRD_TYPE;
                     IN_T6 : in REPORT_TYPE) return natural is

 SUM    : natural := 0;

begin

 for T4 in FOURTH_TYPE'first .. FOURTH_TYPE'last loop

  for T5 in FIFTH_TYPE'first .. FIFTH_TYPE'last loop

   SUM := SUM + RETRIEVE (FIX_1, FIX_2, FIX_3, T4, T5, IN_T6);

  end loop;

 end loop;
```

```
 return SUM;

end GEN_3D_MAT;

--
--
procedure INTERFACE_2D_MAT (T6 : in REPORT_TYPE) is

 type MAT_TYPE is array (FIRST_TYPE, SECOND_TYPE) of natural;
 MAT       : MAT_TYPE;
 T2_NUMBER  : T_NUMBER_TYPE;
 LOOP_NUM   : positive := 1;
 TYPE_TWO   : positive := 2;

begin

 for T1 in FIRST_TYPE'first .. FIRST_TYPE'last loop
  for T2 in SECOND_TYPE'first .. SECOND_TYPE'last loop

     MAT (T1, T2) := GEN_2D_MAT (T1, T2, T6);

  end loop;
 end loop;

 T2_NUMBER := CHECK_TYPE_2;

 if T2_NUMBER = 1 then
  HOW_PRODUCED_HEADING;
 elsif T2_NUMBER = 2 then
  STMT_TYPE_HEADING;
 elsif T2_NUMBER = 3 then
  USAGE_HEADING;
 elsif T2_NUMBER = 4 then
  DEVELOPMENT_STATUS_HEADING;
 end if;

 for T1 in FIRST_TYPE'first .. FIRST_TYPE'last loop
  PRINT_ROW_HEADING (LOOP_NUM);
  for T2 in SECOND_TYPE'first .. SECOND_TYPE'last loop
   put (OUT_FILE_TYPE, MAT (T1, T2), width => 10);
  end loop;
  new_line (OUT_FILE_TYPE, 2);
  LOOP_NUM := LOOP_NUM + 1;
 end loop;

end INTERFACE_2D_MAT;

--
--
procedure INTERFACE_3D_MAT (T6 : in REPORT_TYPE) is
```

```
type MAT_TYPE is array (FIRST_TYPE, SECOND_TYPE) of natural;
MAT       : MAT_TYPE;
LOOP_NUM_3  : positive := 1;
LOOP_NUM_2  : positive := 1;
T3_NUMBER   : T_NUMBER_TYPE;
T2_NUMBER   : T_NUMBER_TYPE;
TYPE_THREE  : positive := 3;

begin

 -- Need to find out the type of the third dimension
 T3_NUMBER := CHECK_TYPE_3;
 T2_NUMBER := CHECK_TYPE_2;

 new_line (OUT_FILE_TYPE, 2);
 -- need to loop through 3rd type here
 for T3 in THIRD_TYPE'first .. THIRD_TYPE'last loop

  -- Creating a two dimensional array with the same third dimesion
  -- staying constant
  for T1 in FIRST_TYPE'first .. FIRST_TYPE'last loop
   for T2 in SECOND_TYPE'first .. SECOND_TYPE'last loop

      MAT (T1, T2) := GEN_3D_MAT (T1, T2, T3, T6);

   end loop;
  end loop;

  -- need to output the thrid dimemsion heading here

  if T3_NUMBER = 1 then
   HOW_PRODUCED_HEADING (LOOP_NUM_3);
  elsif T3_NUMBER = 2 then
   STMT_TYPE_HEADING (LOOP_NUM_3);
  elsif T3_NUMBER = 3 then
   null;
  elsif T3_NUMBER = 4 then
   DEVELOPMENT_STATUS_HEADING (LOOP_NUM_3);
  elsif T3_NUMBER = 5 then
   ORGIN_HEADING (LOOP_NUM_3);
  end if;

  -- Need to output the second dimension as column headings
  -- Finding which type is the second dimension
  if T2_NUMBER = 1 then
   HOW_PRODUCED_HEADING;
  elsif T2_NUMBER = 2 then
   STMT_TYPE_HEADING;
  elsif T2_NUMBER = 3 then
   USAGE_HEADING;
```

```
    elsif T2_NUMBER = 4 then
      DEVELOPMENT_STATUS_HEADING;
    end if;

    -- Printing out the contents of the two dimensional matrix
    for T1 in FIRST_TYPE'first .. FIRST_TYPE'last loop
      PRINT_ROW_HEADING (LOOP_NUM_2);
      for T2 in SECOND_TYPE'first .. SECOND_TYPE'last loop
        put (OUT_FILE_TYPE, MAT (T1, T2), width => 10);
      end loop;
      new_line (OUT_FILE_TYPE, 2);
      LOOP_NUM_2 := LOOP_NUM_2 + 1;
    end loop;

    LOOP_NUM_2 := 1;

    LOOP_NUM_3 := LOOP_NUM_3 + 1;

  end loop;

 end INTERFACE_3D_MAT;


end GENERIC_COUNTS;
```

APPENDIX C. EXTEND SAMPLE INPUT AND OUTPUT

Contents of File list Example

task_package.a

Extended Example Input

```
-- TITLE       : CS 4530 Class Project, Lander
-- AUTHOR      : Kevin J. Walsh and Robert R. Ordonio
-- DATE        : 21 November 1992
-- REVISED     : 22 Nov, 24 Nov,
-- COURSE      : CS 4530, Software Engineering with ADA
-- SYSTEM      : UNIX
-- COMPILER    : Vads6
-- DESCRIPTION : Package contains all the tasks required for the Lander Program

-- 22 Nov (1) Added code to CALCULATE task. Code was to test for the ending
--            conditions, and to be able to exit the loop, and terminate
--            the task when ending condition was found.
-- 24 Nov (1) Reinserted the stop entry call into input task, this will be
--            called by the calculate task after the input is complete
--        (2) Recoded CALCULATE task to perform calculations
-- 27 Nov (1) Integrated keytime code into this package.

---------------------------------------
with DATA_TYPES,
   UTILITY_PKG,
   CURRENT_EXCEPTION,
   TEXT_IO,
   IOCTL,
   TTY,
   OS_FILES;
use  DATA_TYPES,
   UTILITY_PKG,
   TEXT_IO;

package TASK_PACKAGE is

 -- Declaration of local variables used with the package
 sgttyb_buf : tty.sgttyb;
 old_flags  : short_integer;

 -- Instaniation of Enumeration IO to output ROCKET_CONTROL_INPUT variable
 package ROCKET_CONTROL_INPUT_IO is new enumeration_io
(ROCKET_CONTROL_INPUT);
 use ROCKET_CONTROL_INPUT_IO;

 -- Instaniation of Enumeration IO to output the various rocket moter
```

```
  -- variables, rocket positions and fuel capacity.
  package FUEL_IO is new float_io (FUEL);
  use FUEL_IO;

  -- Task will perform all calculations for the program
  task CALCULATE_TASK is
    entry INPUT (ROCKET_DIRECTION : in ROCKET_CONTROL_INPUT);
  end CALCULATE_TASK;

  -- Task will allow the user to input information to the program
  task KEYREAD is
   entry START;
  end KEYREAD;

end TASK_PACKAGE; -- Package specification for task_package

-- Package body for TASK_PACKAGE
package body TASK_PACKAGE is

  -- Declaration of local variables used with package
  CRASH_EXCEPTION,
  MISS_EXCEPTION,
  SKID_EXCEPTION    : exception;


-------------------------------------------------------------
-- Task allows the user to input data to the program.
-- Task will verify input to ensure that input is valid
-------------------------------------------------------------
  task body KEYREAD is

    CHARACTER_INPUT : ROCKET_CONTROL_INPUT;
    CHARACTER_IO    : character;
    DONE            : boolean := FALSE;
    TEST            : natural;

  begin

    select
      accept start;
    or
      terminate;
    end select;

    loop
       -- Get information from the user
       get (CHARACTER_IO);
       -- Determine what the user wants to do
       case CHARACTER_IO is
         when 'u' =>
           test := 0;
```

```
        when 'i' =>
          test := 1;
        when 'o' =>
          test := 2;
        when 'p' =>
          test := 3;
        when 'h' =>
          test := 4;
        when 'j' =>
          test := 5;
        when 'k' =>
          test := 6;
        when 'l' =>
          test := 7;
        when 'v' =>
          test := 8;
        when 'b' =>
          test := 9;
        when 'n' =>
          test := 10;
        when 'm' =>
          test := 11;
        when others =>
          new_line (2);
          put_line (" You have entered the wrong input.  Please try again!");
      end case;
      -- Convert input to rocket control input
      CHARACTER_INPUT := ROCKET_CONTROL_INPUT'VAL(TEST);
      CALCULATE_TASK.INPUT (CHARACTER_INPUT);

    end loop;

    exception
      when others =>
        new_line(2);
        put_line(current_exception.exception_name & " raised in calculate task.");
        new_line(2);
        put_line("Exiting from the input task.  ");
        new_line(2);

  end keyread;

-------------------------------------------------------------
-- Task will perform all calculations for the program.  Task
-- will also check landing and call display procedure to
-- show status of lander information.
-------------------------------------------------------------
  task body CALCULATE_TASK is

    -- Declaration of variables used with the task
    CONTROL_ROCKET : ROCKET_CONTROL_INPUT;
```

```
 TEST        : natural;

begin

 loop

  select
   -- Option to handle user input
   accept INPUT (ROCKET_DIRECTION : in ROCKET_CONTROL_INPUT) do
    CONTROL_ROCKET := ROCKET_DIRECTION;
   end INPUT;
  or
   -- No user input therefore calculate with previous data
   delay 1.0;
    CONTROL_ROCKET := A;
  end select;

  -- Conditional to determine if lander has fuel to manipulate rockets
  if FUEL_LEFT > 0.0 then
   -- Case statement to reset specific rocket input
   case CONTROL_ROCKET is
    when U =>
     POSITIVE_ROCKETS.X := POSITIVE_ROCKETS.X + 1.0;
    when I =>
     if POSITIVE_ROCKETS.X > 0.0 then
      POSITIVE_ROCKETS.X := POSITIVE_ROCKETS.X - 1.0;
     end if;
    when O =>
     NEGATIVE_ROCKETS.X := NEGATIVE_ROCKETS.X + 1.0;
    when P =>
     if NEGATIVE_ROCKETS.X > 0.0 then
      NEGATIVE_ROCKETS.X := NEGATIVE_ROCKETS.X - 1.0;
     end if;
    when H =>
     POSITIVE_ROCKETS.Y := POSITIVE_ROCKETS.Y + 1.0;
    when J =>
     if POSITIVE_ROCKETS.Y > 0.0 then
      POSITIVE_ROCKETS.Y := POSITIVE_ROCKETS.Y - 1.0;
     end if;
    when K =>
     NEGATIVE_ROCKETS.Y := NEGATIVE_ROCKETS.Y + 1.0;
    when L =>
     if NEGATIVE_ROCKETS.Y > 0.0 then
      NEGATIVE_ROCKETS.Y := NEGATIVE_ROCKETS.Y - 1.0;
     end if;
    when V =>
     POSITIVE_ROCKETS.Z := POSITIVE_ROCKETS.Z + 1.0;
    when B =>
     if POSITIVE_ROCKETS.Z > 0.0 then
      POSITIVE_ROCKETS.Z := POSITIVE_ROCKETS.Z - 1.0;
     end if;
```

```
        when N =>
          NEGATIVE_ROCKETS.Z := NEGATIVE_ROCKETS.Z + 1.0;
        when M =>
          if NEGATIVE_ROCKETS.Z > 0.0 then
            NEGATIVE_ROCKETS.Z := NEGATIVE_ROCKETS.Z - 1.0;
          end if;
        when OTHERS =>
          null;
      end case;
    else
      -- FUEL_LEFT = 0 therefore all engines should be at 0.0
      POSITIVE_ROCKETS  := (others => 0.0);
      NEGATIVE_ROCKETS  := (others => 0.0);
    end if;

    -- Calculation for new current position
    CURRENT_POSITIONS.X := CURRENT_POSITIONS.X + DELTA_VECTOR.X;
    CURRENT_POSITIONS.Y := CURRENT_POSITIONS.Y + DELTA_VECTOR.Y;
    CURRENT_POSITIONS.Z := CURRENT_POSITIONS.Z + DELTA_VECTOR.Z;

    -- Calculation for new delta vector
    DELTA_VECTOR.X    := POSITIVE_ROCKETS.X - NEGATIVE_ROCKETS.X;
    DELTA_VECTOR.Y    := POSITIVE_ROCKETS.Y - NEGATIVE_ROCKETS.Y;
    DELTA_VECTOR.Z    := POSITIVE_ROCKETS.Z - NEGATIVE_ROCKETS.Z - 9.8;

    -- Calculation for new fuel left value
    FUEL_LEFT       := FUEL_LEFT - (POSITIVE_ROCKETS.X +
                        POSITIVE_ROCKETS.Y +
                        POSITIVE_ROCKETS.Z +
                        NEGATIVE_ROCKETS.X +
                        NEGATIVE_ROCKETS.Y +
                        NEGATIVE_ROCKETS.Z);

    -- Procedure to display to screen position and info
    DISPLAY (FUEL_LEFT, POSITIVE_ROCKETS, NEGATIVE_ROCKETS,
DELTA_VECTOR,
          START_POSITIONS, CURRENT_POSITIONS, FINAL_POSITIONS);

    -- Test to determine if current position Z lower than final position Z
    if CURRENT_POSITIONS.Z <= FINAL_POSITIONS.Z then
      exit;
    end if;

  end loop;  -- Main initial loop to cycle through calculation task

  -- EVALUATE THE LOCATION AND VELOCITY OF LANDER
  if MISS_DISTANCE (CURRENT_POSITIONS.X,
              FINAL_POSITIONS.X,
              CURRENT_POSITIONS.Y,
              FINAL_POSITIONS.Y) then
    -- LANDING IS CONSIDERED A MISS
```

```
      raise MISS_EXCEPTION;

    elsif SKID_DISTANCE (DELTA_VECTOR.X,
                DELTA_VECTOR.Y) then
      -- LANDING IS CONSIDERED A SKID
      raise SKID_EXCEPTION;

    elsif CRASH_DISTANCE (DELTA_VECTOR.Z) then
      -- LANDING IS CONSIDERED A CRASH
      raise CRASH_EXCEPTION;

    else
      -- LANDING IS CONSIDERED A SUCCESS!!!!
      new_line(2);
      put_line("CONGRATULATIONS.  Successful landing accomplished. ");
      new_line(2);
      abort KEYREAD;
    end if;

    exception
      when MISS_EXCEPTION =>
        new_line (2);
        put_line ("  SORRY, BUT THE LANDING IS CONSIDERED A MISS");
        put_line ("  Calculations stopped, program aborted. ");
        new_line (2);
        abort KEYREAD;
      when SKID_EXCEPTION =>
        new_line (2);
        put_line ("  SORRY, BUT THE LANDING IS CONSIDERED A SKID");
        put_line ("  Calculations stopped, program aborted. ");
        new_line (2);
        abort KEYREAD;
      when CRASH_EXCEPTION =>
        new_line (2);
        put_line ("   SORRY, BUT THE LANDING IS CONSIDERED A CRASH");
        new_line (2);
        abort KEYREAD;

  end CALCULATE_TASK;

end TASK_PACKAGE; -- Package body for task_package
```

Extended Example Output

REPORT A

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 193
Estimated: 0

	Total Includes	Total Excludes	Individual totals
Statement type			
When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1 Executables	Precedence => 1	XXXX	157
2 Nonexecutables			
3 Declarations	2	XXXX	36
4 Compiler Directives	3	XXXX	0
5 Comments			
6 On their own lines	4	XXXX	0
7 On lines with source code	5	XXXX	0
8 Banners and nonblank spacers	6	XXXX	0
9 Blank (empty) comments	7	XXXX	0
10 Blank lines	8	XXXX	0

How Produced

1 Programmed	XXXX	193
2 Generated with source code generators	XXXX	0
3 Converted with automated translators	XXXX	0
4 Copied or reused without change	XXXX	0
5 Modified	XXXX	0
6 Removed	XXXX	0

Orgin

1 New Work: no prior existence	XXXX	193

2 Prior work: taken or adapted from
3 A previous version, build, or release XXXX 0
4 Commercial, off the shelf software COTS), other than libraries XXXX 0
5 Government furnished software (GFS), other than reuse libraries XXXX 0
6 Another product XXXX 0
7 A vendor-supplied language support library (unmodified) XXXX 0
8 A vedor-supplied operating system or utility (unmodified) XXXX 0
9 A local or modified language support library or operating system XXXX 0
10 Other commercial library XXXX 0
11 A reuse library (software designed for reuse) XXXX 0
12 Other software component or library XXXX 0

Usage

1 In or as part of the primary product XXXX 193
2 External to or in support of the primary product XXXX 0

Development Status

1 Estimated or planned XXXX 0
2 Designed XXXX 0
3 Coded XXXX 0
4 Unit tests completed XXXX 0
5 Integrated into components XXXX 0
6 Test readiness review completed XXXX 0
7 Software (CSCI) tests completed XXXX 0
8 System tests completed XXXX 193

REPORT B

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 193
Estimated: 0

		Total Includes	Total Excludes	Individual totals
Statement type				
	When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1	Executables	Precedence => 1	XXXX	157
2	Nonexecutables			
3	Declarations	2	XXXX	36
4	Compiler Directives	3	XXXX	0
5	Comments			
6	On their own lines	4	XXXX	0
7	On lines with source code	5	XXXX	0
8	Banners and nonblank spacers	6	XXXX	0
9	Blank (empty) comments	7	XXXX	0
10	Blank lines	8	XXXX	0
How Produced				
1	Programmed		XXXX	193
2	Generated with source code generators		XXXX	0
3	Converted with automated translators		XXXX	0
4	Copied or reused without change		XXXX	0
5	Modified		XXXX	0
6	Removed		XXXX	0
Orgin				
1	New Work: no prior existence		XXXX	193
2	Prior work: taken or adapted from			
3	A previous version, build, or release		XXXX	0
4	Commercial, off the shelf software COTS), other than libraries		XXXX	0

5	Government furnished software (GFS), other than reuse libraries	XXXX	0
6	Another product	XXXX	0
7	A vendor-supplied language support library (unmodified)	XXXX	0
8	A vedor-supplied operating system or utility (unmodified)	XXXX	0
9	A local or modified language support library or operating system	XXXX	0
10	Other commercial library	XXXX	0
11	A reuse library (software designed for reuse)	XXXX	0
12	Other software component or library	XXXX	0

Usage

1	In or as part of the primary product	XXXX	193
2	External to or in support of the primary product	XXXX	0

Development Status

1	Estimated or planned	XXXX	0
2	Designed	XXXX	0
3	Coded	XXXX	0
4	Unit tests completed	XXXX	0
5	Integrated into components	XXXX	0
6	Test readiness review completed	XXXX	0
7	Software (CSCI) tests completed	XXXX	0
8	System tests completed	XXXX	193

	Programmed	Generated	Converted	Copied	Modified	Removed
Estimated or planned	0	0	0	0	0	0
Designed	0	0	0	0	0	0
Coded	0	0	0	0	0	0
Unit tests completed	0	0	0	0	0	0
Integrated into components	0	0	0	0	0	0
Test readiness review completed	0	0	0	0	0	0

Software (CSCI) tests completed	0	0	0	0	0	0
System tests completed	193	0	0	0	0	0

REPORT C

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 240
Estimated: 0

	Total Includes	Total Excludes	Individual totals
Statement type			
When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1 Executables	Precedence => 1	XXXX	157
2 Nonexecutables			
3 Declarations	2	XXXX	36
4 Compiler Directives	3	XXXX	0
5 Comments			
6 On their own lines	4	XXXX	47
7 On lines with source code	5	XXXX	0
8 Banners and nonblank spacers	6	XXXX	0
9 Blank (empty) comments	7	XXXX	0
10 Blank lines	8	XXXX	0
How Produced			
1 Programmed		XXXX	240
2 Generated with source code generators		XXXX	0
3 Converted with automated translators		XXXX	0
4 Copied or reused without change		XXXX	0
5 Modified		XXXX	0
6 Removed		XXXX	0
Orgin			
1 New Work: no prior existence		XXXX	240
2 Prior work: taken or adapted from			
3 A previous version, build, or release		XXXX	0
4 Commercial, off the shelf software			

	COTS), other than libraries	XXXX	0
5	Government furnished software (GFS), other than reuse libraries	XXXX	0
6	Another product	XXXX	0
7	A vendor-supplied language support library (unmodified)	XXXX	0
8	A vedor-supplied operating system or utility (unmodified)	XXXX	0
9	A local or modified language support library or operating system	XXXX	0
10	Other commercial library	XXXX	0
11	A reuse library (software designed for reuse)	XXXX	0
12	Other software component or library	XXXX	0

Usage

1	In or as part of the primary product	XXXX	240
2	External to or in support of the primary product	XXXX	0

Development Status

1	Estimated or planned	XXXX	0
2	Designed	XXXX	0
3	Coded	XXXX	0
4	Unit tests completed	XXXX	0
5	Integrated into components	XXXX	0
6	Test readiness review completed	XXXX	0
7	Software (CSCI) tests completed	XXXX	0
8	System tests completed	XXXX	240

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	157	0	0	0	0	0
Declarations	36	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	47	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0

Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

REPORT D

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 193
Estimated: 0

		Total Includes	Total Excludes	Individual totals
Statement type				
	When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1	Executables	Precedence => 1	XXXX	157
2	Nonexecutables			
3	Declarations	2	XXXX	36
4	Compiler Directives	3	XXXX	0
5	Comments			
6	On their own lines	4	XXXX	0
7	On lines with source code	5	XXXX	0
8	Banners and nonblank spacers	6	XXXX	0
9	Blank (empty) comments	7	XXXX	0
10	Blank lines	8	XXXX	0
How Produced				
1	Programmed		XXXX	193
2	Generated with source code generators		XXXX	0
3	Converted with automated translators		XXXX	0
4	Copied or reused without change		XXXX	0
5	Modified		XXXX	0
6	Removed		XXXX	0
Orgin				
1	New Work: no prior existence		XXXX	193
2	Prior work: taken or adapted from			
3	A previous version, build, or release		XXXX	0
4	Commercial, off the shelf software COTS), other than libraries		XXXX	0

5	Government furnished software (GFS), other than reuse libraries	XXXX	0
6	Another product	XXXX	0
7	A vendor-supplied language support library (unmodified)	XXXX	0
8	A vedor-supplied operating system or utility (unmodified)	XXXX	0
9	A local or modified language support library or operating system	XXXX	0
10	Other commercial library	XXXX	0
11	A reuse library (software designed for reuse)	XXXX	0
12	Other software component or library	XXXX	0

Usage

1	In or as part of the primary product	XXXX	193
2	External to or in support of the primary product	XXXX	0

Development Status

1	Estimated or planned	XXXX	0
2	Designed	XXXX	0
3	Coded	XXXX	0
4	Unit tests completed	XXXX	0
5	Integrated into components	XXXX	0
6	Test readiness review completed	XXXX	0
7	Software (CSCI) tests completed	XXXX	0
8	System tests completed	XXXX	193

	Programmed	Generated	Converted	Copied	Modified	Removed
New Work: no prior existence	193	0	0	0	0	0
A previos version, build, or release	0	0	0	0	0	0
COTS	0	0	0	0	0	0
GFS	0	0	0	0	0	0
Another product	0	0	0	0	0	0
A vendor supplied language support library	0	0	0	0	0	0

A vendor-supplied operating system or utility	0	0	0	0	0	0
A local or modified language support library or operating system	0	0	0	0	0	0
Other commercial library	0	0	0	0	0	0
A reuse library (software designed for reuse	0	0	0	0	0	0
Other software component or library	0	0	0	0	0	0

REPORT E

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 240
Estimated: 0

	Total Includes	Total Excludes	Individual totals
Statement type			
When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1 Executables	Precedence => 1	XXXX	157
2 Nonexecutables			
3 Declarations	2	XXXX	36
4 Compiler Directives	3	XXXX	0
5 Comments			
6 On their own lines	4	XXXX	47
7 On lines with source code	5	XXXX	0
8 Banners and nonblank spacers	6	XXXX	0
9 Blank (empty) comments	7	XXXX	0
10 Blank lines	8	XXXX	0
How Produced			
1 Programmed		XXXX	240
2 Generated with source code generators		XXXX	0
3 Converted with automated translators		XXXX	0
4 Copied or reused without change		XXXX	0
5 Modified		XXXX	0
6 Removed		XXXX	0
Orgin			
1 New Work: no prior existence		XXXX	240
2 Prior work: taken or adapted from			
3 A previous version, build, or release		XXXX	0
4 Commercial, off the shelf software COTS), other than libraries		XXXX	0

5	Government furnished software (GFS), other than reuse libraries	XXXX	0
6	Another product	XXXX	0
7	A vendor-supplied language support library (unmodified)	XXXX	0
8	A vedor-supplied operating system or utility (unmodified)	XXXX	0
9	A local or modified language support library or operating system	XXXX	0
10	Other commercial library	XXXX	0
11	A reuse library (software designed for reuse)	XXXX	0
12	Other software component or library	XXXX	0

Usage

1	In or as part of the primary product	XXXX	240
2	External to or in support of the primary product	XXXX	0

Development Status

1	Estimated or planned	XXXX	0
2	Designed	XXXX	0
3	Coded	XXXX	0
4	Unit tests completed	XXXX	0
5	Integrated into components	XXXX	0
6	Test readiness review completed	XXXX	0
7	Software (CSCI) tests completed	XXXX	0
8	System tests completed	XXXX	240

Orgin.New work

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	157	0	0	0	0	0
Declarations	36	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	47	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0

Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.Previos version

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.COTS

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on						

lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.GFS

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.Annother product

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on						

their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.A vendor supplied language support library

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler dir-ectives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.A vendor supplied operating system or utility

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler dir-						

ectives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.A local or modified language support library

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler dir-ectives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.Other commercial library

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0

Declarations	0	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.A reuse library

	Programmed	Generated	Converted	Copied	Modified	Removed
Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

Orgin.Other software component or library

	Programmed	Generated	Converted	Copied	Modified	Removed

Executable	0	0	0	0	0	0
Declarations	0	0	0	0	0	0
Compiler directives	0	0	0	0	0	0
Comments on their own line	0	0	0	0	0	0
Comments on lines with source code	0	0	0	0	0	0
Banner and non-blank spacers	0	0	0	0	0	0
Blank (empty) comments	0	0	0	0	0	0
Blank lines	0	0	0	0	0	0

REPORT F

Report Name: Thesis example
File List used: example
Requested by: Kevin J. Walsh

Measured as: Physical source lines

Delivered as: Delivered as source

Counted: 284
Estimated: 0

	Total Includes	Total Excludes	Individual totals
Statement type			
When a line or statement contains more than one type, classify it as the type with the highest precedence.			
1 Executables	Precedence => 1	XXXX	157
2 Nonexecutables			
3 Declarations	2	XXXX	36
4 Compiler Directives	3	XXXX	0
5 Comments			
6 On their own lines	4	XXXX	47
7 On lines with source code	5	XXXX	0
8 Banners and nonblank spacers	6	XXXX	5
9 Blank (empty) comments	7	XXXX	0
10 Blank lines	8	XXXX	39
How Produced			
1 Programmed		XXXX	284
2 Generated with source code generators		XXXX	0
3 Converted with automated translators		XXXX	0
4 Copied or reused without change		XXXX	0
5 Modified		XXXX	0
6 Removed		XXXX	0
Orgin			
1 New Work: no prior existence		XXXX	284
2 Prior work: taken or adapted from			
3 A previous version, build, or release		XXXX	0
4 Commercial, off the shelf software COTS), other than libraries		XXXX	0

5	Government furnished software (GFS), other than reuse libraries	XXXX	0
6	Another product	XXXX	0
7	A vendor-supplied language support library (unmodified)	XXXX	0
8	A vedor-supplied operating system or utility (unmodified)	XXXX	0
9	A local or modified language support library or operating system	XXXX	0
10	Other commercial library	XXXX	0
11	A reuse library (software designed for reuse)	XXXX	0
12	Other software component or library	XXXX	0

Usage

1	In or as part of the primary product	XXXX	284
2	External to or in support of the primary product	XXXX	0

Development Status

1	Estimated or planned	XXXX	0
2	Designed	XXXX	0
3	Coded	XXXX	0
4	Unit tests completed	XXXX	0
5	Integrated into components	XXXX	0
6	Test readiness review completed	XXXX	0
7	Software (CSCI) tests completed	XXXX	0
8	System tests completed	XXXX	284

LIST OF REFERENCES

[BEI 90] Beizer, Boris, *Software Testing Techniques*, 2d ed., pp 213-242, Van Nostrand Reinhold, 1990.

[BER 90] Berzins, V., Luqi, *Software Engineering with Abstractions*, pp 1-21, Addison-Wesley Publishing Company, 1990

[CSC 92] Communications System Center/Software Department, Tinker Air Force Base, Oklahoma, "*The Source Code Line Counter Program*," T. Goff, pp. 1-8, 17 December, 1992.

[NAS 90] McCabe, Tom, "A Complexity Measure," *IEEE Transactions Software Engineer*, pp 308-320, December 1976.

[NAS 90] National Aeronautics and Space Administration, Goddard Space Flight Center, *Transportable Applications Environment Plus User Interface Developer's Guide*, v. 5.1, pp 1-265, April 1991.

[NGU 88] Nguyen, T., Forester, K., *ALEX - An Ada Lexical Analysis Generator, Version 1.0*, Arcadia Environment Research Project, Department of Information and Computer Science, University of California, Irvine, 1988.

[SLI 87] Set Laboratories, Inc., *PC-METRIC (PASCAL), pp 1-1 to 6-5, 1987.*

[SEL 90] Self, J., *AFLEX - A fast lexical analyzer generator for Ada, Version 1.1*, Arcadia Environment Research Project, Department of Information and Computer Science, University of California, Irvine, 1 September 1990.

[SEI 93] Software Engineering Institute, Software Engineering Symposium, *The Business of Software Engineering: The Competitive Edge, The SEI Measurements Checklist--User Experience*, August 1993.

[SEI-A 92] Software Engineering Institute, Technical Report 19, *Software Measurement for DoD Systems: Recommendations for Initial Core Measures*, Carleton, A., D., and others, pp. 1-68, September 1992.

[SEI-B 92] Software Engineering Institute, Technical Report 20, *Software Size Measurement*, Park, R. E., pp. 1-167, September 1992

[SEI-C 92] Software Engineering Institute, Technical Report 21, *Software Effort & Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information*, Goethert, W., B., Bailey, E., K., and Busby, M., B., pp 1-10, September 1992.

[SEI-D 92] Software Engineering Institute, Technical Report 22, *Software Quality Measurement: A Framework for Counting Problems and Defects*, Florac, W. A., pp 1-20, September 1992

[SSD 90] Software Systems Design, Inc., *ADADL User's Manual*, Release 5.0, pp 1-130, July 1990.

[SUN 90] Sun Microsystems, Inc, *SunOS Reference Manual*, Revision A of 27 March, 1990.

[TAB 88] Taback, D., Deepak, T., *AYACC - Users Manual, Version 1.0*, Arcadia Environment Research Project, Department of Information and Computer Science, University of California, Irvine, 1988.

[WAR 90] Warner Books Inc., *Webster's New World Dictionary*, pp 38, 236, August 1990.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-5002	2
Dr. Timothy J. Shimeall Computer Science Department Code CS/Sm Naval Postgraduate School Monterey, CA 93943-5118	4
MAJ David Gaitros Computer Science Department Code CS/Ga Naval Postrgraduate School Monterey, CA 93943-5118	1
Dr. Ted Lewis Computer Science Department Code CS/Lt Naval Postgraduate School Monterey, CA 93943-5118	1
MAJ Kevin J. Walsh 18 Carty Ave. Fort Monmouth, NJ 07703	3
Ms. Anita Carleton Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213-3890	1
Dr. Marshall Potter NISMC-O3 Bldg. 166 Washington Navy Yard Washington, DC 20374-5070	1

432 709

DUDLE
NAVAL
MON
ARY
TE SCHOOL
3101

AMERICAN BINDERY
2
6-84

DUDLEY KNOX LIBRARY
3 2768 00019479 9